NASA-CR-192175

EXPERT SYSTEM VERIFICATION AND VALIDATION STUDY

FINAL REPORT

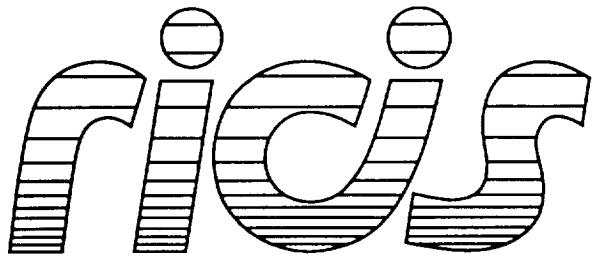
Scott W. French
David Hamilton

International Business Machines Corporation

August 1992

Cooperative Agreement NCC 9-16
Research Activity No. AI.16

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division



Research Institute for Computing and Information Systems
University of Houston-Clear Lake

TECHNICAL REPORT

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

EXPERT SYSTEM VERIFICATION AND VALIDATION STUDY

FINAL REPORT

RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Scott W. French and David Hamilton of the International Business Machines Corporation. Dr. T. F. Leibfried, Jr. served as RICIS research coordinator.

Funding was provided by the Information Technology Division, Information Systems Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA research coordinator for this activity was Christopher Culbert, Chief, Software Technology Branch, Information Technology Division, Information Systems Directorate, NASA/JSC.

RICIS Contract 069 — Final Report

Document Number 5

August 28, 1992

Scott W. French
David O. Hamilton

IBM Corporation
3700 Bay Area Blvd.
Houston, TX 77058

Preface

This report satisfies deliverable number 5 of RICIS contract #069. The purpose is to document results of the four Expert Systems Verification and Validation workshops taught during the period of March 1992 to August 1992.

Background

Five workshops on Verification and Validation (V&V) of Expert Systems (ES) where taught during this recent period of performance. Two key activities, previously performed under this contract, supported these recent workshops.

1. Survey of state-of-the-practice of V&V of ES
2. Development of workshop material and first class

The next two section describe these activities in more detail.

Survey

The first activity involved performing an extensive survey of ES developers in order to answer several questions regarding the state-of-the-practice in V&V of ES. These questions related to the amount and type of V&V done and the successfulness of this V&V[1]. The answers to these questions led us to two primary conclusions:

1. The state-of-the-practice in V&V of ES needs to be improved. This conclusion came from the lack of V&V techniques used, the fact that many systems did not meet expectations, and other results that indicated a relatively unstructured approach to V&V. This does not necessarily mean that the systems being developed were of poor quality or that the developers were not trying; in fact, many project spent a considerable amount of time doing V&V, in some cases up to 80% of the effort was spent on V&V activities. The results only indicate that many systems did not meet expectations and the V&V approach appeared inadequate in most cases.
2. A great improvement in ES V&V could be achieved by using existing V&V techniques and methods. That is, although ES V&V is still an active research area with many unsolved problems, there are still many existing methods and techniques that are just not being used. Furthermore, many of these methods and techniques are ones that are being applied to conventional (i.e., non ES) software. It seemed as though the large body of knowledge about general V&V was not being applied to ES.

These conclusions led us to believe that a class on V&V, with a special emphasis toward ES, would be of great value in improving the state-of-the-practice in ES V&V. Specifically, we wanted to inform developers about:

1. basic V&V concepts
2. the most useful V&V methods and techniques
3. differences between ES and conventional software
4. V&V technques specifically for ES

Additionally, we hoped to provide some hands-on experience with these techniques.

Workshop Development

The next key activity involved developing an intensive hands-on workshop in V&V of ES. This activity

[1] The full results of this survey has been delivered in a previous report under this contract.

involved surveying a large number of V&V techniques, conventional as well as ES specific ones.[2]

In addition to explaining the techniques, we showed how each technique could be applied on a sample problem. References were included in the workshop material, and cross referenced to techniques, so that students would know where to go to find additional information about each technique.

In addition to teaching specific techniques, we included an extensive amount of material on V&V concepts and how to develop a V&V plan for an ES project. We felt this material was necessary so that developers would be prepared to develop an orderly and structured approach to V&V. That is, they would have a process that supported the use of the specific techniques.

Finally, to provide hands-on experience, we developed a set of case study exercises. These exercises were to provide an opportunity for the students to apply all the material (concepts, techniques, and planning material) to a realistic problem.

First Class and Review of Workshop

Through our previous work, we had made contacts with a number of leading researchers in the area of ES V&V. We sent copies of our workshop material to these researchers and solicited their feedback and we updated the material based on their feedback. It is worth noting that we received many positive comments about the workshop material and our plans to teach it to ES developers. Based on this review, we felt confident that we had a solid and comprehensive set of course material.

The next review step was to present an overview of the material to a group of people knowledgeable in ES but not experts in ES V&V. This review gave us an opportunity to find out how well we could communicate the information to practitioners. A number of concerns were raised at this review. We addressed these concerns primarily by adding material to explain the role and purpose of V&V in ES development. This review was at the end of the previous phase of this contract (February, 1992) and constituted the first class.

[2] In particular, we capitalized on previous work done by Lance Miller of Science Applications International Corporation in support of a contract to the Electrical Power Research Institute and the Nuclear Regulatory Commission.

Teaching the Workshop

The first full class was taught in March of 1992 to a select group of ES developers and software V&V professionals. The students were arranged by the NASA JSC Software Technology Branch (STB). This class lasted three full days.

At the end of this first class, we, along with Chris Culbert and Bebe Ly of the STB conducted an interactive discussion with the students on the value of the material and suggestions for improvement. It was felt that the material was very valuable and did need to be taught to others. For example, one student said that they had been doing V&V for six years (and had to learn things on the job), but still learned some new things in the class; "I just wish I had had this class six years ago" , they said.

We also learned many ways the material could be improved and the way it was presented could be improved. We learned that more time was needed for exercises and the order of presentation needed to be revised. Based on the recommendations and further analysis of the course material, an improved version of the workshop was produced. The new version had a new outline that allowed us to interleave lectures and the case study exercises, rather than having the students do all the exercises at the end of the class. We also added some videotaped demonstrations to further break up the lectures and make the class more interesting. This new version required four days to teach, instead of three.

To solicit students for additional classes, we developed a flyer that the STB circulated for us. We had scheduled three additional classes and received enough responses to hold each class.[3]

The additional classes did not indicate a need to modify the existing material, though some new material was added. The new material was in the form of worksheets that walked the students, step by step, through applying several of the more advanced techniques. Examples of the kind of work performed by the students is in "Attachment B" on page 11.

Computer-Based Training Prototype

To support the advertisement of the workshop, IBM prototyped a multimedia introduction to the course. This presentation is centered around a problem situation that occured during the Apollo 11 lunar landing. This problem is an intriguing example that can be used to discuss many V&V concepts and motivate people to attend the workshop.

The development of this prototype resulted in a new item being added to the contract. This item involves providing assistance to the STB in the development of an expanded multimedia computer-based training tool that illustrates concepts from the workshop.

Management Overview

One major goal of the workshop was to "sell" developers on the importance of V&V activities. In general we were very successful. Most students indicated a belief that the ideas taught were the "right way to go." However, they also indicated that teaching their managers these same concepts would increase the likelihood of their success in applying them.

[3] Refer to "Attachment C" on page 13 for a complete list of workshop attendees.

Therefore, we developed a two hour management overview [4]. The management overview covers the same basic concepts of V&V without focusing on techniques. The idea is to tell managers why V&V is beneficial and how they can help their ES project developers as they try to apply the ideas taught in the workshop. At this point, however, the management overview has not been taught.

[4] Refer to "Attachment D" on page 15 for a copy of the management overview.

Results

Overall, ***the workshops were very successful.*** The evaluation scores shown in "Results of Student Evaluations" on page 7 reflect that success. We also received many positive comments about the value of the material. However, it is still too early to detect any long-term impact of the class on current V&V practice. Activities are underway, however, to begin assessing this long-term impact.

As with any successful job, however, there is always room for improvement. The remainder of this report will focus primarily on areas where the workshop could be improved. These areas of improvement have been derived from informal discussions with the students, student evaluations and instructor observations.

Student Profile

In order to put student comments/evaluations into perspective, it is helpful to consider the profile of the students that attended each of the workshops. This section describes that student profile and how it impacted the successes and shortcomings of the workshop.

The ES V&V workshop was taught five times during the period of March 1992 to August 1992. The workshop in March was substantially different from the others held from May to August. This was the case for several reasons: the class material discussed fewer techniques, the class was shorter (three days instead of four), and the students were "hand-picked" to attend and give valuable feedback on the first class. Since the students were "hand-picked" for the March class, the attendance was very good (a consistent twenty-one each day of the class).

Many helpful suggestions were implemented as a result of the March workshop. In addition, a flyer was distributed by the Software Technology Branch (STB) to advertise the next four workshops (May through August). People wishing to attend the workshop were asked to either call IBM or the STB to register for the workshop. As we learned later, more time should have been spent screening people requesting to attend the class. The flyer indicated that the workshop would be of primary interest to those currently working ES problems. However, the background of the students actually attending the workshop reflected a broader scope of interest. The following statements reflect comments made to the instructor by students when describing their reason for attending the workshop:

- "I am here because my manager signed me up"
- "I am here to learn about ES; I do not care about V&V."
- "I am here to learn about V&V; I do not care about ES."
- "I am here to learn how to use ES to do V&V"
- "I do not know why I am here; I am a programmer. I do not do testing."

Unfortunately, the difficulty in finding the right students for the workshop meant that student attendance was not as good as the March workshop. With the exception of the June workshop, each of the workshops started with 15 or more students (our goal was to have at least 20). Attendance, however, steadily dwindled each day to the point where the workshop usually ended with only one-half to two-thirds of the original students still in attendance. It should be noted that many who stopped attending did so because of demands on the job. Yet, many left because they thought the workshop was going to be something that it was not. We have attempted to address this issue by asking each student to fill out a questionaire, indicating what information they would most like to get out of the class. We then use this information to focus the lecture time on information of the most interest. However, the best solution in the future would be to better communicate the goals of the class to each prospective student, so we can be sure that the class meets the students needs and expectations.

An example of this latter point is evident in the number of students who had no idea what rule-base programming was all about. In the "Basic Concepts" section of the workshop, an example of a rule-based program and a procedural program are contrasted to illustrate key impacts of "AI" languages on V&V. In the March workshop the rule-based example was written in CLIPS. Roughly one-third of those students knew nothing about CLIPS. Therefore, we spent about an hour explaining the basics of CLIPS. To avoid this problem in future classes, the CLIPS examples were modified to use a English-like pseudo language (still rule-based). The idea was to remove the requirement to know CLIPS syntax. This only partially solved the problem, because anywhere from one-third to two-thirds of the workshop students in the next workshops did not know anything about rule-based programming or any AI language (in the case of the June workshop, there was only one student in the class that knew anything about rule-based programming). This increased the difficulty in contrasting ES and procedural V&V issues.

Some of the dissatisfaction with the material on techniques may be due to the state-of-the-art in ES V&V. As with software in general, there is no "silver bullet" as some would like to find. Instead, there are only techniques that require discipline and skill to apply. Students that worked hard to apply the tehniques on the team exercises generally had a more positive response to the workshop. However, some students (as evidenced by some of the responses in "Attachment B" on page 11) expended minimal effort on the exercises. The reasons for this are not clear but may be due to the discipline required to apply many of the techniques. For example, one of the better techniques for evaluating rule-based programs is to generate "connectivity" matrices. This technique is good because it is relatively easy and could easily be automated with off-the-shelf matrix operations routines. But it is somewhat tedious and only finds anomalies that must be analyzed to determine if they are faults or not. Thus, it requires discipline and definitely is not a "silver bullet" solution. One student indicated that they had learned about matrices in school and did not want to have to use them anymore. We recognize that some of the techniques are somewhat unpleasant and not "fun" to use. It is for this reason that we are recommending that a significant effort be made to select the best techniques and then automate them.

We attempted to teach about techniques and also about developing a V&V approach. The information required to develop a V&V approach is much less than what is required to master the techniques. So we were able to thoroughly cover V&V planning but, could only introduce each technique. To thoroughly cover each technique would have required an order of magnitude more time, something on the order of a one-semester college course. We attempted to overcome this shortcoming by providing extensive references which were cross referenced to techniques. We also provided a set of worksheets that would help a student follow each technique in a step by step fashion. Still, many students would have rather covered fewer techniques, but covered them more in depth. We feel the best way to resolve this issue is to teach a class that covers one or two development methodologies that support V&V. The methodology would be composed of a few techniques that work together well and address a wide range of V&V.

Students who are technical project leads found the class very exciting and rewarding. This was because they did not need as much detailed information. Instead, they just needed a general overview of issues that they need to consider in leading their projects. Many of these students explicitly said, "I am going to start using this information on my project." Others indicated that this class will be very helpful as a "reference" when "planning for V&V" on their project.

On the other hand, students who came to get specific help on specific problems found the experience less rewarding because the workshop was not geared for that. For example, one particular student attended the class wishing to receive specific help in verifying a G2 rule-base. Rather than focusing on specific problems like this one, the workshop tried to address broader information about V&V concepts, issues and techniques[5]. In some cases, though, the information was too broad, because many students had a stronger background in the concepts of V&V than we anticipated. This was not necessarily a big problem (i.e., some

[5] The survey results indicated that most ES projects were built by people with little training in V&V (i.e., they were engineers, flight controllers, etc. – not programmers).

enjoyed the review of V&V concepts), but reflects on why some of the evaluation comments indicated there should be less emphasis on "basic concepts."

The workshop was definitely geared to students whose jobs are in the technical area of systems development. However, it soon became clear that many approaches being advocated in the class would be difficult for the student to do without significant commitment from their management. Many students brought this concern to the attention of the instructors. A good example of this is found in the team exercise solutions of "Attachment B" on page 11. One group indicated that they would not use the technique of inspections as part of their V&V approach. Their reason? "My management will not pay for inspections." This answer was given despite a thorough presentation on the benefits of inspections[6].

In summary, a better job of "screening" students and a better job of communicating the course intent should solve many of these problems. The objective of each workshop was clear: provide information on V&V and encourage people to use V&V techniques on ES (i.e., V&V of an ES can be done, despite what a student may have heard to the contrary). The purpose was not to make the student an expert on using each technique or to teach expert system programming techniques/languages. To this end, the class, at least so far, has been very successful.

Results of Student Evaluations

Rating (1 = High, 5 = Low)	*Category*
1.9	Quality of course material
1.6	Effectiveness of Instructors
2.1	Depth of course content
1.7	Degree to which the course met its stated objective
1.9	Effectiveness of the delivery method
2.0	Relevance of the course to my job requirements
1.9	Confidence in my ability to apply the course content to my job
2.0	Course exercises
2.3	Length of course

Student Comments/Suggestions

The following items are a condensation of comments received from students at the bottom of the class evaluation form (see "Attachment A" on page 9).

- Too much standard s/w engineering basics; not enough knowledge-based related material
- Less emphasis on basics and more in depth exploration of techniques/guidelines
- Course should probably be longer to allow a bit more detail on techniques
- More abstract examples
- More examples of what an ES is. This hard for one to pick up with no previous experience in ES.
- Preferred a shorter class focussing on specific techniques in great detail
- Split the workshop into two pieces: basic concepts and techniques
- Course is more of an overview - more detail on techniques is needed

[6] Students learned that, on average, inspections find roughly sixty percent of all errors.

- Course was too long and the team exercises too drawn out
- The "References" at the end of each section make the course excellent and very useful
- Improve the quality of video presentations
- Present instructor "solutions" to team exercises on the last day
- Better student attendance would have helped when applying techniques to team exercises
- Course should have had a real-world problem (using G2) with terminals for each student where students could apply specific analysis techniques

Observations/Recommendations of the Instructors

The following are suggestions for improving on the work done with the ES V&V workshop:

- Do a better job of "screening" people wishing to take the class
- Learn about G2 and have examples on how to apply V&V techniques to a G2 rule-base
- Advertise the workshop as two two-day workshops: "Basic Concepts" and "Techniques/Guidelines"
- Spend more time on techniques with improved examples and more detailed discussion
- Teach a management version of this workshop
- Fund future work in automating some of the better techniques; many of the techniques lack automation and therefore will be minimally used
- Improve the use of video during the workshop. For example, a video could be made illustrating (via some "role-play") a knowledge acquisition process and application of knowledge correctness techniques to that process.
- Develop a "corollary" workshop on how to build verifiable ES (using techniques such as "cleanroom")
- Spend some time discussing what to do when you already have the system done and that system was not built using a V&V approach

Attachment A

CLASS EVALUATION

Your input helps us improve course offerings. **Date:** _____

Course Name: _____

Instructor: _____

Your Name (Optional): _____

(circle one response)

1 = Very Satisfied 2 = Satisfied 3 = Neither Satisfied nor Dissatisfied 4 = Dissatisfied 5 = Very Dissatisfied

	OVERALL
1 2 3 4 5	***How satisfied are you with this educational experience?***

What influenced your answer? (circle one response per factor)

1 = Very Positively 2 = Positively 3 = No Influence 4 = Negatively 5 = Very Negatively

	Course Content/Delivery Factors
1 2 3 4 5	a. Degree to which the course met its stated objectives
1 2 3 4 5	b. Depth of the course content
1 2 3 4 5	c. Length of the course
1 2 3 4 5	d. Effectiveness of the instructor(s)
1 2 3 4 5	e. Course exercises/labs
1 2 3 4 5	f. Effectiveness of the delivery method (e.g., Classroom, Satellite, etc.)
1 2 3 4 5	g. Quality of the course materials
	Job Application Factors
1 2 3 4 5	h. Availability of the course when needed
1 2 3 4 5	i. Relevance of the course content to my job requirements
1 2 3 4 5	j. Confidence in my ability to apply the course content to my job
	Administrative/Environmental Factors
1 2 3 4 5	k. Enrollment process
1 2 3 4 5	l. Class administrative services
1 2 3 4 5	m. Education facilities (e.g., classroom climate, equipment, etc.)
1 2 3 4 5	n. Travel (when applicable)
1 2 3 4 5	o. Lodging (when applicable)
	Other factors
1 2 3 4 5	p. Please specify _____

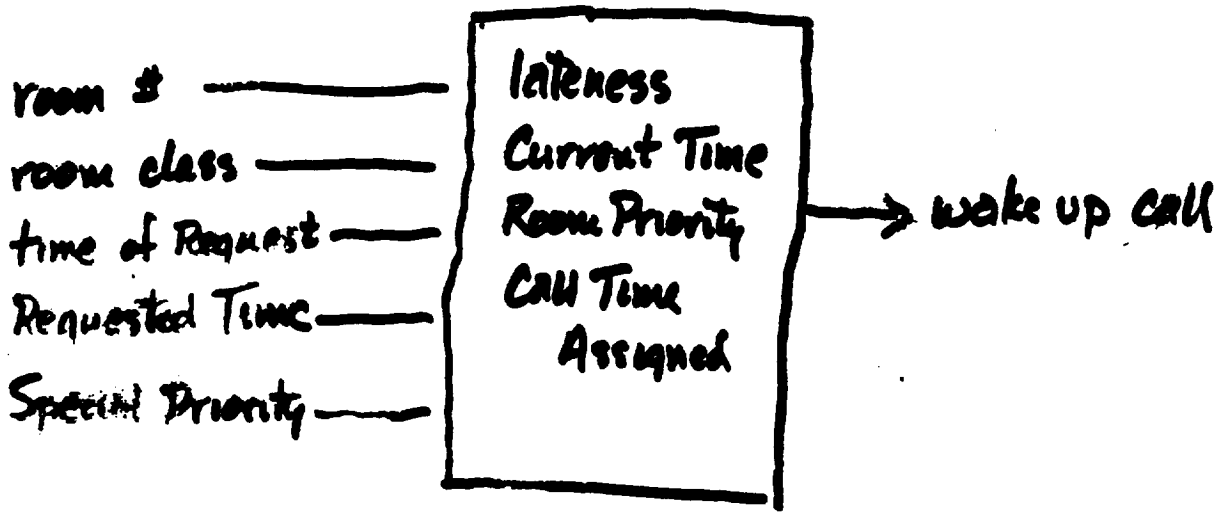
Please provide any additional comments if appropriate. _____

Attachment B

Team #1

Handout #7: Exercises on General Techniques

1. Define the "black box" view for your system.



2. Identify key terms from the problem description.

room #
room Class
current Time
call Time
desired Time
~~request Time~~ Time of Req.
spec. priority

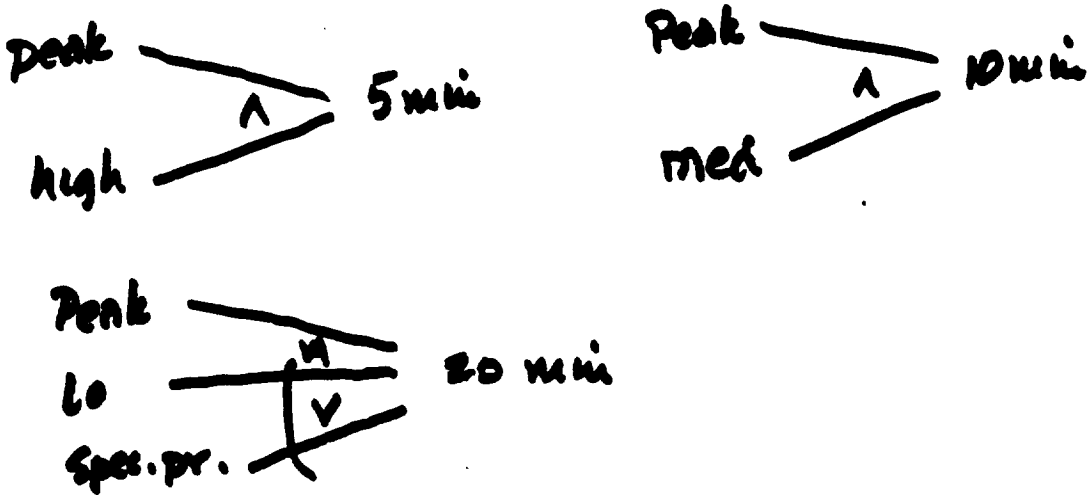
3. Which of the following techniques would you use? Explain your answer.

- Prototyping — proof of concept, test user interface
- Competing Designs
- Independent V&V
- Inspections — catch 40% of errors early.

4. Do a very high level specification for your system using one of the following techniques:

- Decision Table
- Cause-Effect Graph
- State Diagram

Early



Late

Early req —— higher prior = # min

call is late —— pr = 6 × # min

late > 20 min
Spec. Prior
higher prior. in same class

late > 20 min
Spec prior.
higher priority in all classes

Handout #8: Exercises on System Test Techniques

1. Define 1 or more "realistic" test cases for your team exercise.

avg # rooms make avg # requests

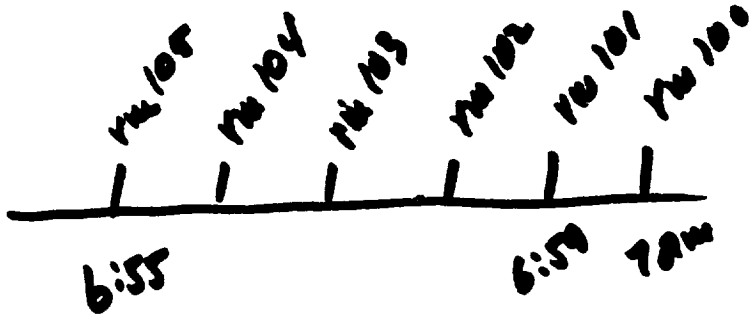
see if wake up calls made correctly

2. Define some attributes of your system. Define 1 or more test cases based on those attributes.

?

3. Define 1 or more test cases that do "boundary value" testing.

Have 6 hi class rooms request reserve 7am



4. Define 1 or more test cases that "stress" test the system.

- Have 7 hi class rooms request 7am
- Have 11 med class rooms req. 7am
- Have 7 hi class + 11 med class req. 7am

5. Define the external interfaces to your system. Define 1 or more test cases to test those interfaces.

1. reservationist I/O interface
2. dial up machine

---

1. call reservationist w/ request - watch I/O
2. program wake up call - see it happen

6. Define 1 or more test cases to test the system's performance.

Ask expert for normal peak load & during busy season.

Run simulation - shouldn't fall up computer watch hardware load

7. For each question, indicate how the results of each test case will be analyzed (i.e., how you will know the answer is correct).

1. realistic 2. altrn. 3. boundary - syst. gives expected results

4. stress - syst. survives

5. user I/o - acceptable response

8. Did the problem description provide enough detail to adequately perform the tests from questions 1-6?

No - didn't answer what to do if wakeup call not answered

9. Develop a "certification" test for your system.

10. Identify system "disasters" (i.e., things that should not happen). Explain how you will test your system for these "disasters".

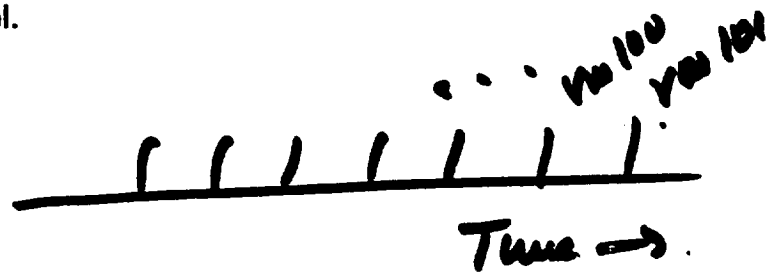
power failure

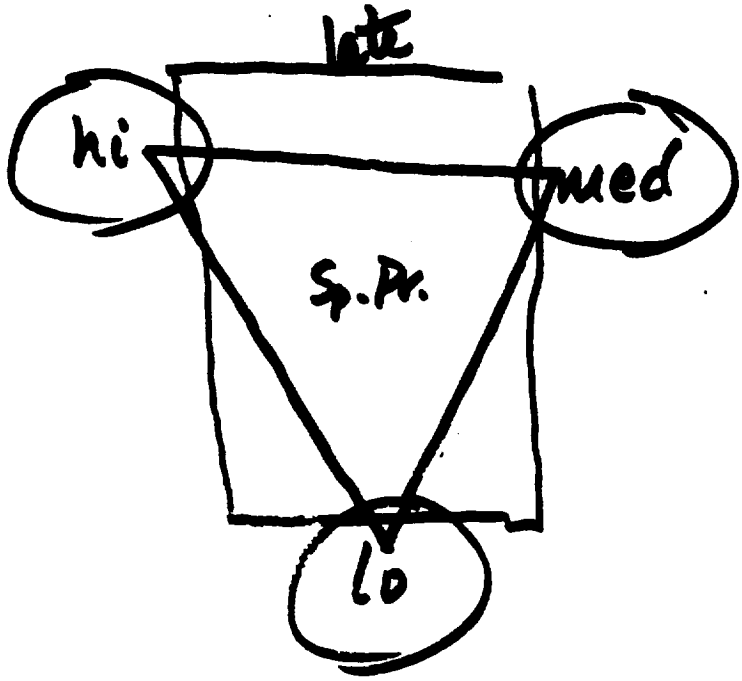
phone syst. out of order

11. Will your project need the aid of an expert (provide rationale)? If so, indicate the kind of expert required and the type of analysis to be performed.

experienced operator needed to verify day to day operation

12. Define 1 or more models to aid in your understanding of the system. Document each model.





Handout #9: Exercises on Unit/Integration Test Techniques

1. Pick an implementation approach for your problem. Based on this choice, would you use:
   - Coverage techniques — Path Coverage make sure ea. rule fires in corr. order
   - Interprocedural data-flow analysis

Implem. Approach

I. log calls (user id)

II. schedule calls (rule for each calls, run Class, prior)

III. Make wake up calls (user d/p-call mark in, rules abt wakeup calls

IV. Reschedule calls (bitmap rules, run Class, ...)

2. Identify "part" of the system that may impact reliability (HINT: you may have to define what reliability is). Define 1 or more test cases to test those "parts".

Wake Up Calls must be made w/in acceptable time frame

Clock - run a pgm to req. time -
- Check agnst watch

3. Document 1 or more expected sequences of actions for your system.

- rm 100 calls in & req. wake up @ 7am
- pgm schedule wake up call for 7am
- pgm makes " " @ 7am

Add rm 200 (need clean) req. 7am
pgm shud resched to 6:59

4. Is "prototype evaluation" appropriate for your problem? What about mutation testing? Provide rationale.

prototyping for user I/O

mutation testing for sched & resched callTime.

5. Exchange your work with another team. Study the problem. Ask yourself the following:

- Does their implementation match the problem?
- Are there any "holes" or inconsistencies in their descriptions?
- Did they pick the right techniques for their implementation approach?

Handout #10: Exercises on Static Test Techniques

1. Identify and define at least 1 "object" in your system (remember, objects consist of both data and operations on that data).

A room has rm #
rm class
desired WakeUp Time
Time wake up was requested
assigned Call Time
spec. prior

Operation: Assign a call Time

pre: desired WakeUp / class / spec Room / Time R

if { all rooms } callTime != newRoom.desiredTi

Then newRm.callTime = "

post: call Time is assigned a value

v1.

2. Write a pre-condition and a post-condition for each operation on the object.

r2 Opn: call a room @ assigned call Time

7:02a 7:02a

pre: assigned Call Time, curr. Time

if curTime = call Time

Then call room is confirmed

post: put call in que

---

r3 Opn: Make calls from call que

pre: class, sp. pr, time Req, Cur Time, call To
desired Time

post: call in ASAP

3. Describe any general properties your "object" must satisfy. Discuss how you would analyze your "object"'s implementation to "prove" those properties are always satisfied.

Assigned call Time < cur Time + 20

Class must be (lo, med, hi)

Spec. pr must be (True, False)

4. Pick at least one operation and defined some rules that implement its specificiation.

see above #1

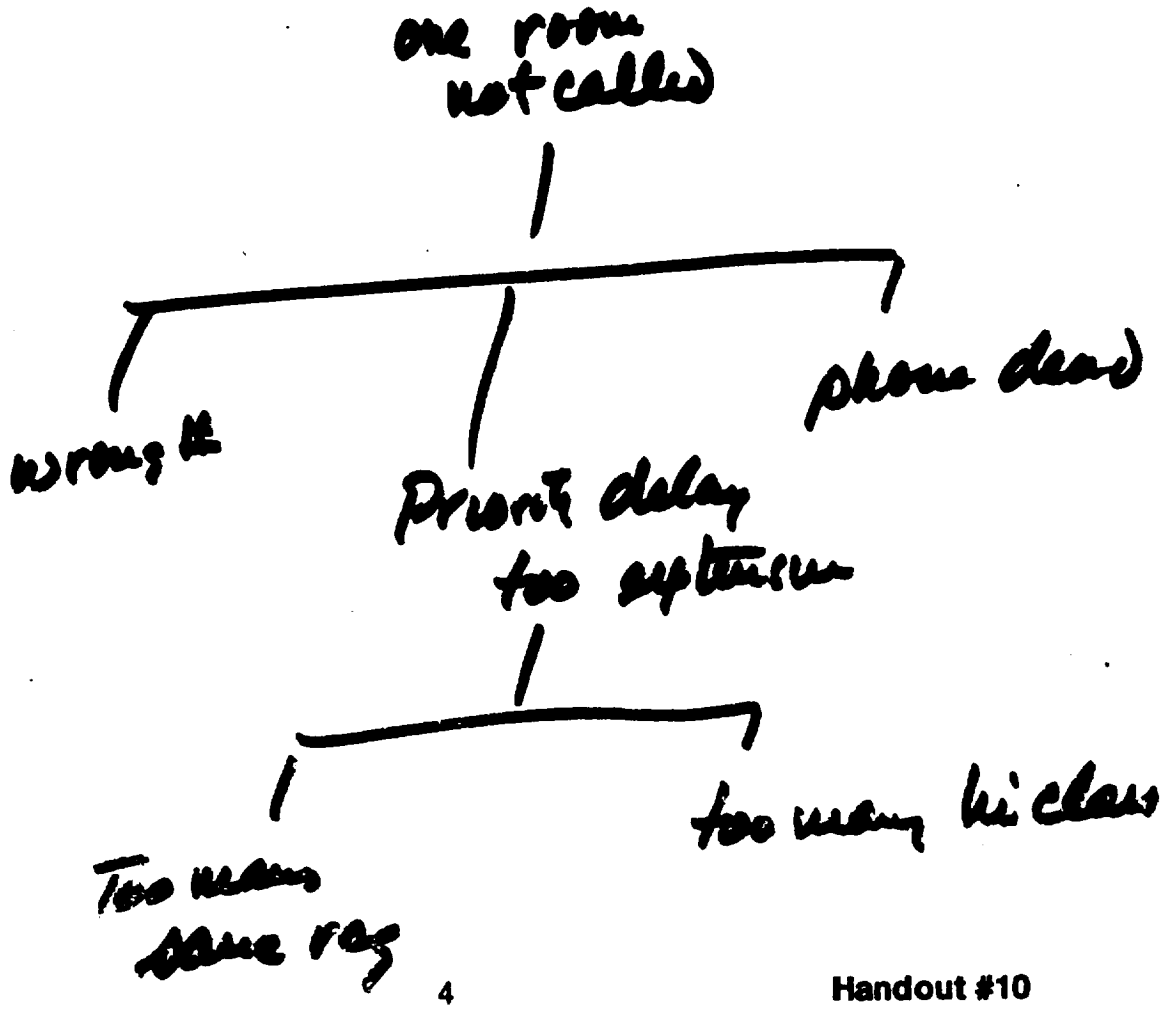
5. Select one of the following techniques for analyzing these rules. Explain your answer.

- Petri Nets
- Directed Graphs
- Connectivity Matrices

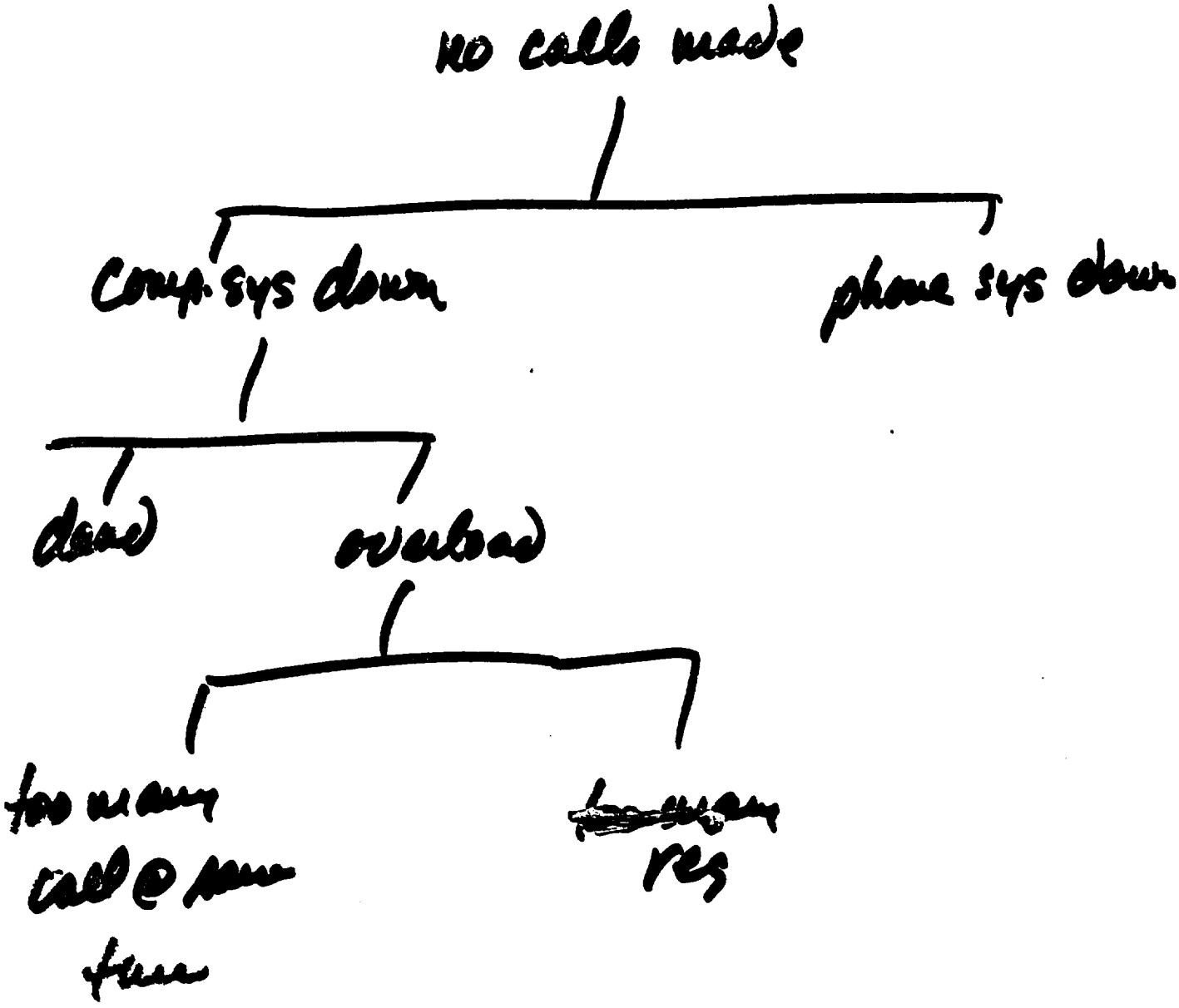
	r_1	r_2	r_3
r_1	0	1	1
r_2	1	1	1
r_3	1	1	1

r_1 doesn't trigger r_1
r_1 does trigger r_2 !
r_2 " " r_1, r_3
r_3 " " " "

6. Identify 1 "hazard" in your system. Build a fault tree for that "hazard".



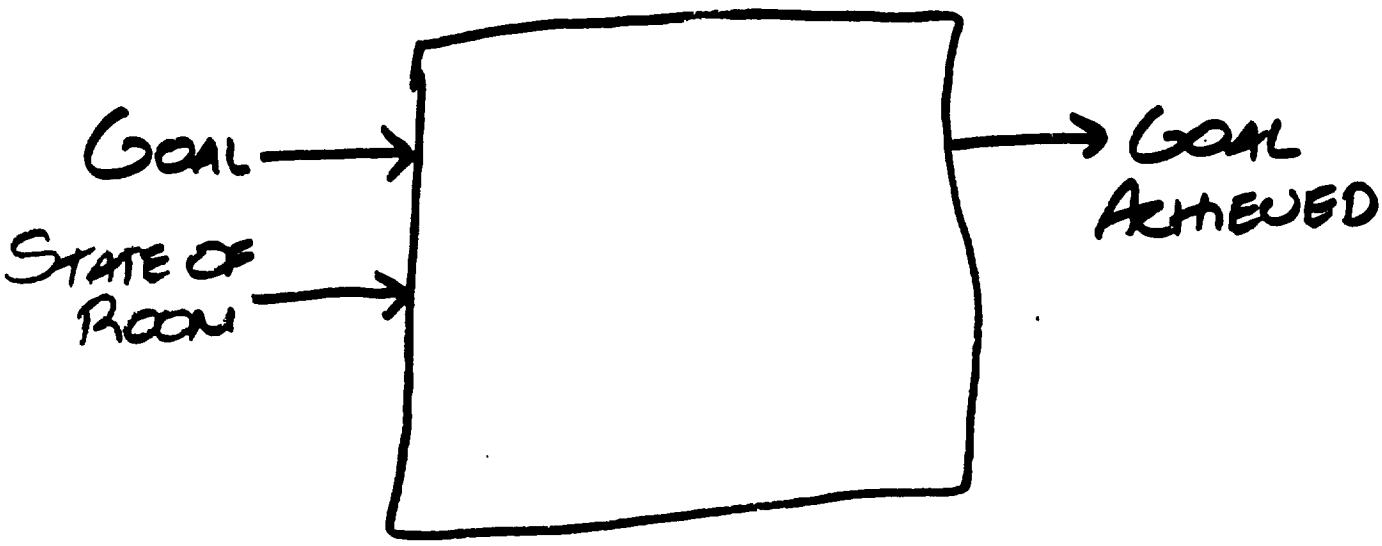
7. Idenfity 1 "fault" in your system. Build a fault tree for that "fault".



Team #2

Handout #7: Exercises on General Techniques

1. Define the "black box" view for your system.



2. Identify key terms from the problem description.

OBJECTS (MONKEY, KEY, PILLOW)
GOALS (UNLOCK, EAT, CLIMB)
ATTRIBUTES (COLOR, LOCATION)
ACTIONS (HOLD, WALK, JUMP)
INITIAL CONDITIONS

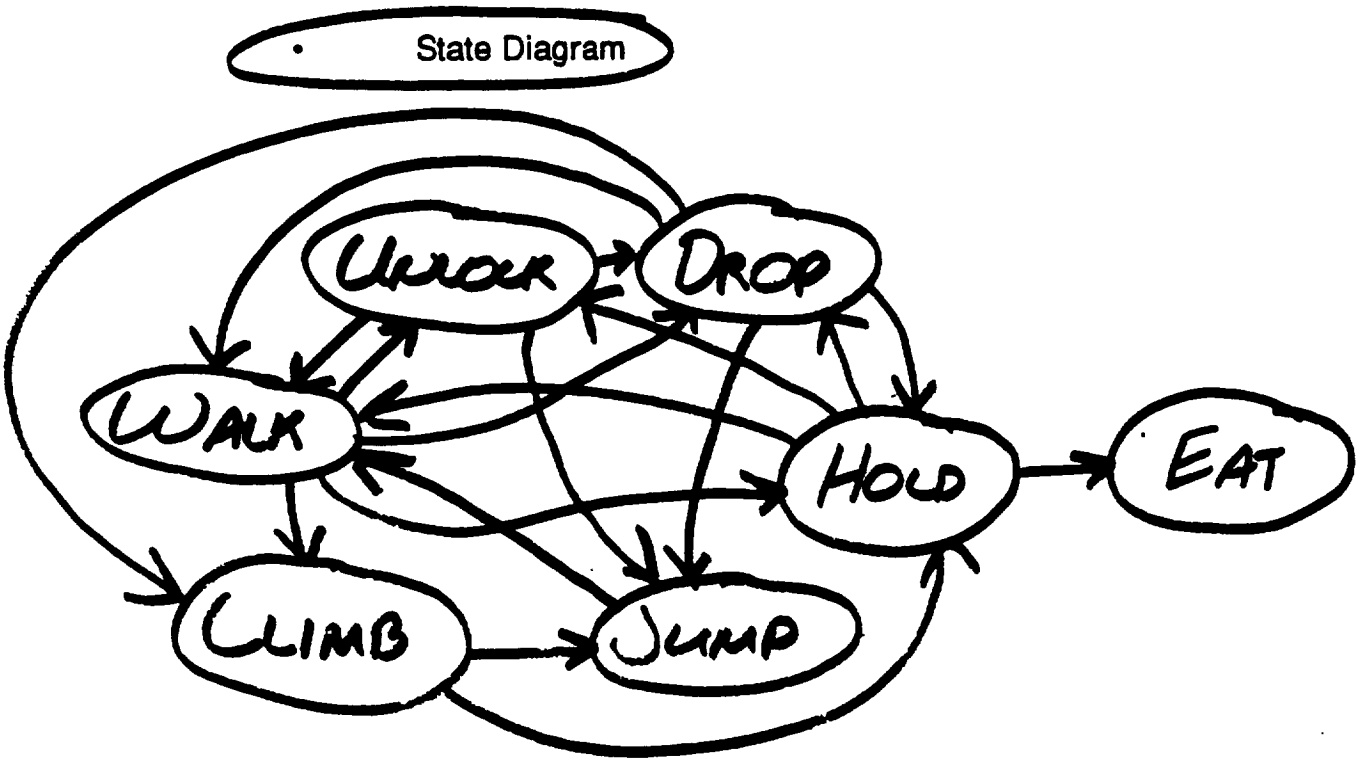
GOALS AND ACTIONS ARE CLOSELY RELATED.

3. Which of the following techniques would you use? Explain your answer.

- Prototyping - TEST CONCEPTS AND BASELI KNOWLEDGE REPRESENTA
- Competing Designs - WIDER RANGE OF IMPLEMENTATION IDEAS
- Independent V&V
- Inspections - NO EXPERT OR CRITICALITY TO ACHIEVE OPTIMUM PO (ARE WE TRYING TO MODE THE 'AVERAGE' MONKEY OR A GENIUS?)

4. Do a very high level specification for your system using one of the following techniques:

- Decision Table
- Cause-Effect Graph
- State Diagram



Handout #8: Exercises on System Test Techniques

1. Define 1 or more "realistic" test cases for your team exercise.

ANY SCENARIO THAT INVOLVES A 'LEGAL' SITUATION.

GIVEN A SQUEAKLE INITIAL STATE, ASSERT A GOAL AND OBSERVE THE ACTIONS.

2. Define some attributes of your system. Define 1 or more test cases based on those attributes.

LOCATION - CAN THE MONKEY GET FROM ONE LOCATION TO ANOTHER.

3. Define 1 or more test cases that do "boundary value" testing.

PLACE OBSTACLES ON THE ROOM BOUNDARY TO MAKE SURE THE MONKEY STAYS IN THE ROOM.

4. Define 1 or more test cases that "stress" test the system.

SEE ABOVE.

COMPLEX SITUATIONS (MULTIPLE ITEMS ON THE KEY)

5. Define the external interfaces to your system. Define 1 or more test cases to test those interfaces.

NONE, OUTSIDE OF ESTABLISHING THE INITIAL CONDITION.

6. Define 1 or more test cases to test the system's performance.

WIDE VARIETY OF COMPLEXITY, RANGING FROM BANANAS NEXT TO MONKEY TO ALL OBSTACLES BETWEEN MONKEY AND BANANAS.

7. For each question, indicate how the results of each test case will be analyzed (i.e., how you will know the answer is correct).

PRIMARILY - ACHIEVEMENT OF GOAL.
SECONDARY - PATH TAKEN.

8. Did the problem description provide enough detail to adequately perform the tests from questions 1-6?

YES!

9. Develop a "certification" test for your system.

FULL STOMACH!

10. Identify system "disasters" (i.e., things that should not happen). Explain how you will test your system for these "disasters".

KEY LOCKED IN THE CABINET
THAT IT OPENS, OR
KEY NOT IN ROOM.

(ANY INVALID INITIAL STATE.

LIMIT ON MAXIMUM NUMBER
OF ACTIONS TAKEN.

11. Will your project need the aid of an expert (provide rationale)? If so, indicate the kind of expert required and the type of analysis to be performed.

NO.

12. Define 1 or more models to aid in your understanding of the system. Document each model.

ACTIONS OF A PET OR YOUNG CHILD.
ROBOT WITH LIMITED KNOWLEDGE.

Handout #9: Exercises on Unit/Integration Test Techniques

1. Pick an implementation approach for your problem. Based on this choice, would you use:

Coverage techniques

- Interprocedural data-flow analysis

1) DEFINE ACTIONS
2) DEFINE STATES
3) DEFINE TRANSITIONS BETWEEN STAT
4) DETERMINE KNOWLEDGE REPRESENT
5) COMBINE TRANSITIONS TO ACHIEVE MORE COMPLEX GOALS.

2. Identify "part" of the system that may impact reliability (HINT: you may have to define what reliability is). Define 1 or more test cases to test those "parts".

ALL PARTS ARE CRITICAL TO ACHIEVE THE GOAL.

RESOLUTION OF EACH ACTION MUST BE FOLLOWED BY SELECTION OF AN APPROPRIATE NEXT ACTI

ANY TEST CASE INVOLVING PAIRS OF ACTIONS WILL HELP VERIFY THIS CAPABILITY.

3. Document 1 or more expected sequences of actions for your system.

MONKEY PICKS UP, HOLDS AND EATS BANANAS.

MONKEY PICKS UP AND HOLDS KEY, WALKS TO AND UNLOCKS CHEST.

4. Is "prototype evaluation" appropriate for your problem? What about mutation testing? Provide rationale.

PROTOTYPE EVALUATION - YES TO DETERMINE SOUNDNESS OF IMPLEMENTATION APPROACH AND KNOWLEDGE REPRESENTA

MUTATION TESTING

#10

Handout #9

2. Write a pre-condition and a post-condition for each operation on the object.

OPERATION	PRE	POST
HOLD	GOAL TO HOLD MONKEY AT SAME LOCATION MONKEY HOLDING NOTHING .	NEW GOAL MONKEY HOLDING KEY .

Handout #10: Exercises on Static Test Techniques

1. Identify and define at least 1 "object" in your system (remember, objects consist of both data and operations on that data).

KEY

DATA - LOCATION, COLOR, WEIGHT
POSITION

OPERATION - HOLD, DROP, UNLOCK,

3. Describe any general properties your "object" must satisfy. Discuss how you would analyze your "object"'s implementation to "prove" those properties are always satisfied.

CAN BE HELD
UNLOCKS A PREDETERMINED
CABINET.

STATE DIAGRAM.

4. Pick at least one operation and defined some rules that implement its specificiation.

HOLD - SEE PRE AND POST
CONDITION.

PRECEDING PAGE BLANK NOT FILMED

5. Select one of the following techniques for analyzing these rules. Explain your answer.

- Petri Nets
- Directed Graphs
- Connectivity Matrices

6. Identify 1 "hazard" in your system. Build a fault tree for that "hazard".

KEY LOCKED IN THE CABINET THAT IT UNLOCKS.

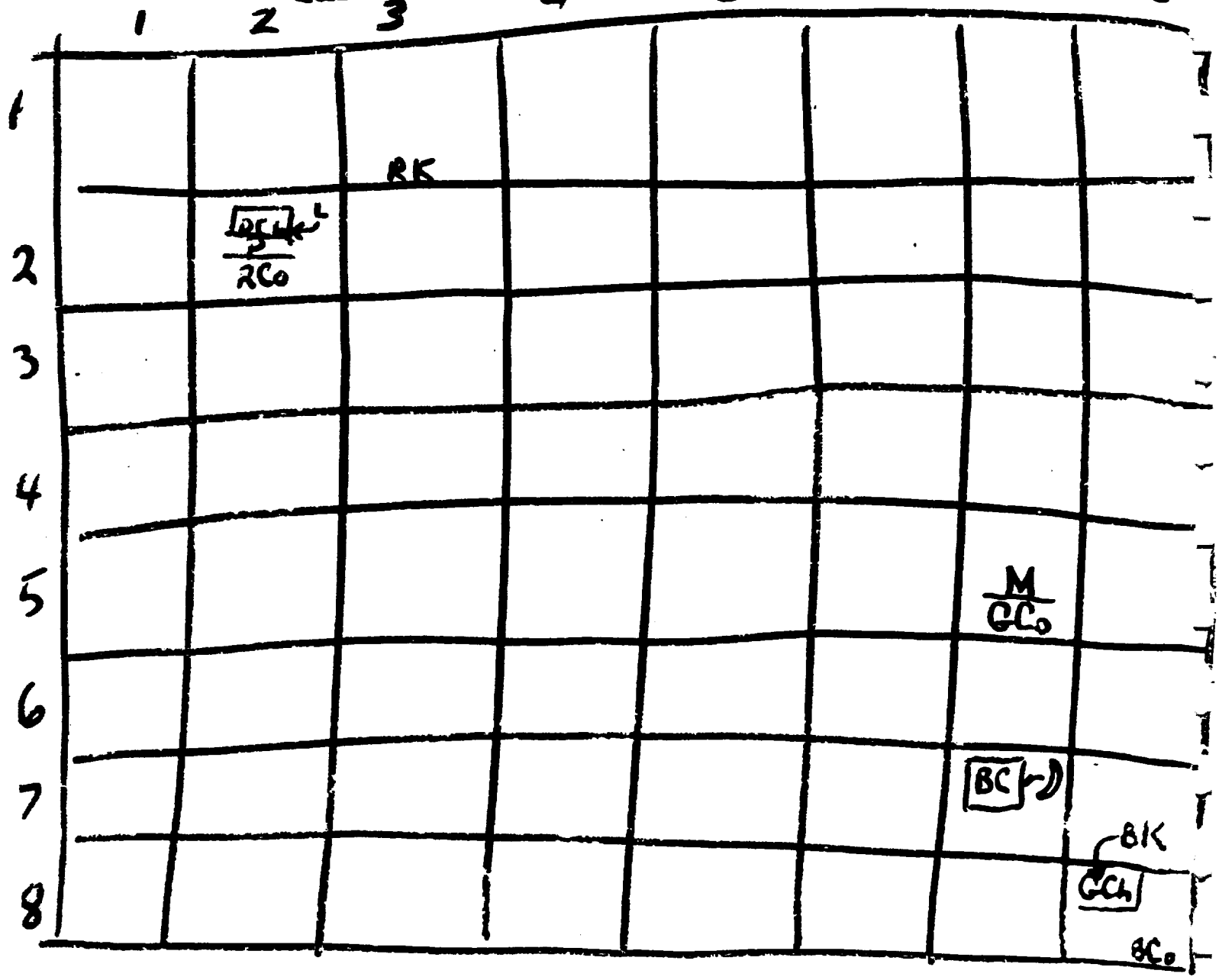
Monkeys and Bananas

7. ~~Identify a "fault" in your system. Build a fault tree for that "fault".~~

RCh Red Chest
P Pillow
RCo Red Couch
RK Red Key
L Ladder

M - Monkey
GCo - Green Couch
BCh - Blue Chest
B - Banana

GCh - Gr... Che...
BCo - Blu... Cou...
BK - Blue



Handout #11: Exercises on Guidelines

1. Determine whether the recommended approach fits your problem. Identify additional issues that need to be considered.

YES
NONE

2. Generate a detailed development plan for your problem. Try to include specific milestones and how they will be achieved.

ANALYZE- STUDY PROBLEM DESCRIPTION, ANALYZE HOLES.

DESIGN - DEFINE ACTIONS, STATES, TRANSITION BETWEEN STATES, KNOWLEDGE REPRESENTATION, MODEL OF ACTIONS, STATES AND TRANSITIONS

PROTOTYPE - IMPLEMENT REPRESENTATION, COMPLETING SIMPLE TASKS. REFINEMENT OF REPRESENTATI

IMPLEMENT - COMBINE SIMPLE TASKS TO FORM MORE COMPLEX TASKS AND ACHIEVE GOAL. ADD CONTRO STRUCTURE TO DETERMINE NE GOAL AND USER INTERFACE

3. Define specific development increments. Update your plan to reflect those increments.

SEE PREVIOUS SLIDE.

4. Consider the test cases you have selected so far. Are there any other kinds of testing you need to do? When will you know when to stop testing?

STOP TESTING WHEN.

1) EACH VALID ACTION CAN BE PERFORMED. (UNIT TESTING)
2) EACH PAIR OF ACTIONS CAN BE PERFORMED AND ARE CORRECTLY DETERMINED (INTEGRATION TESTING).
3) VALID/INVALID INITIAL CONDITIONS TO SEE THAT GOAL IS ACHIEVED & SYSTEM DEGRADES GRACEFULLY (SYSTEM TESTING).

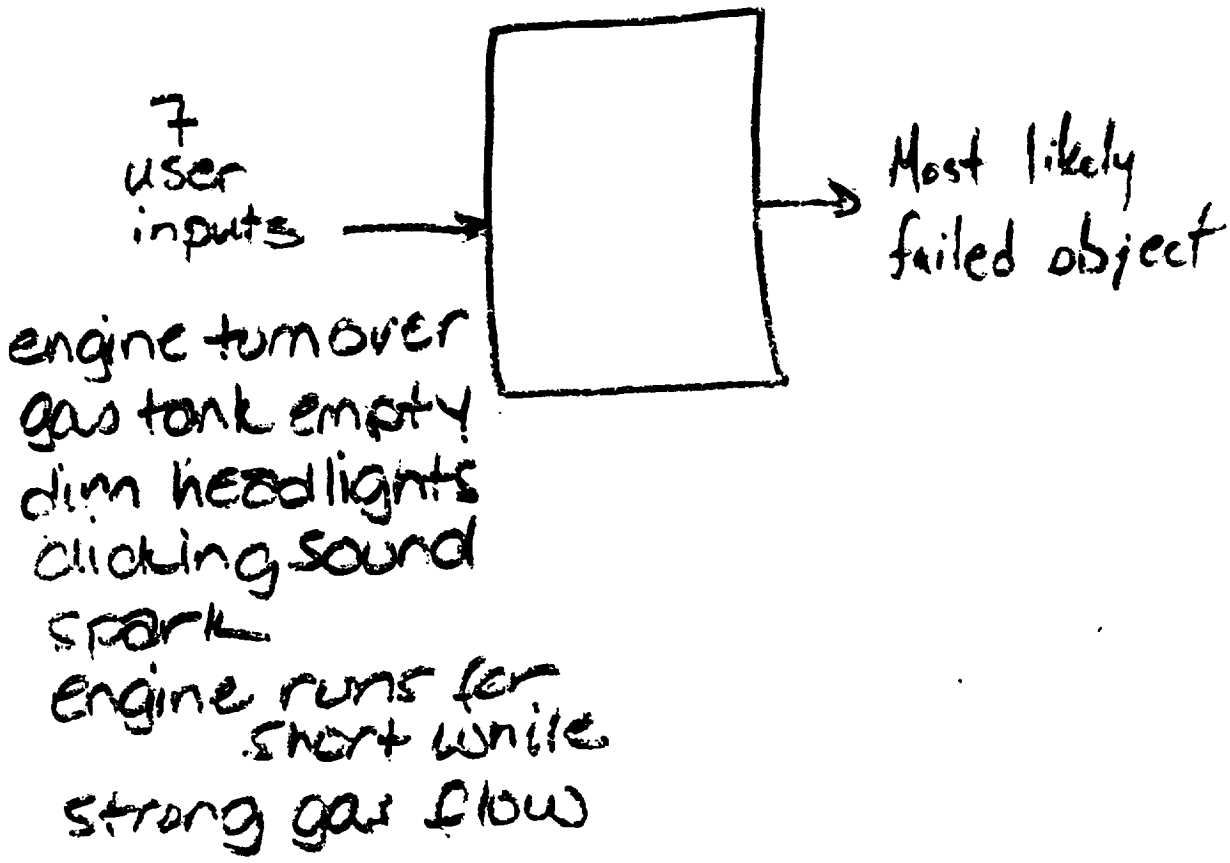
5. Build a high-level requirements outline for your system. How well does the original problem definition map to your outline?

FIND BANANAS
GET BANANAS
EAT BANANAS
TAKE A NAP

Cam #3

Handout #7: Exercises on General Techniques

1. Define the "black box" view for your system.



2. Identify key terms from the problem description.

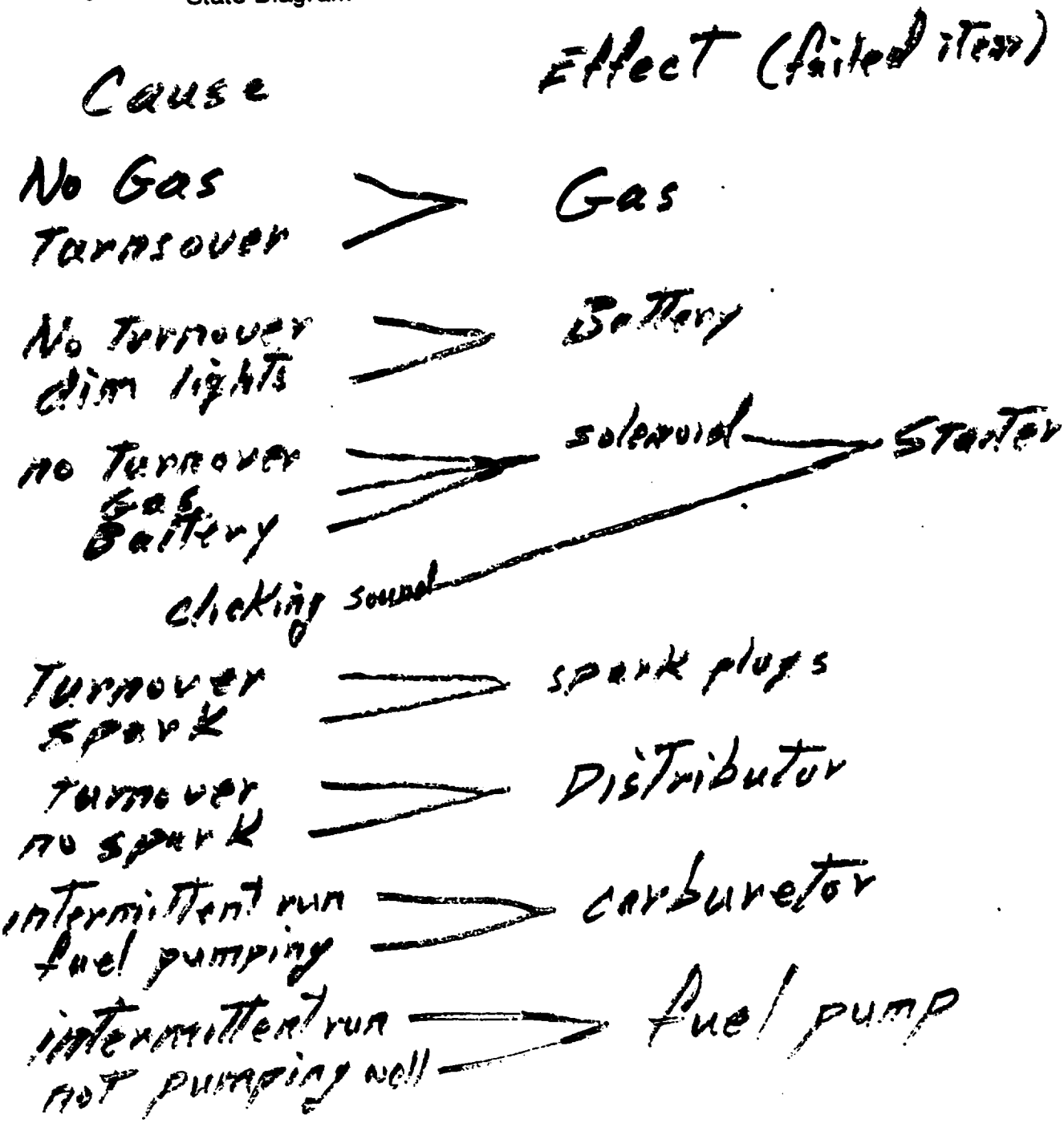
- Gas gauge empty
- clicking sound
- Gas strongly squirts
- Check spark
- Headlights not shining brightly
- Intermittent running
- Engine turns over / doesn't turn over

3. Which of the following techniques would you use? Explain your answer.

- Prototyping – user interface
- Competing Designs
- Independent V&V – test requirements are met
- Inspections – code

4. Do a very high level specification for your system using one of the following techniques:

- Decision Table
- Cause-Effect Graph
- State Diagram



Handout #8: Exercises on System Test Techniques

1. Define 1 or more "realistic" test cases for your team exercise.

TC 1 - Gas tank / battery

User I/F inputs for questions:

Does engine turn over?

Yes - gas

If yes - is gas gauge reading empty? No - Next

If no - do headlights shine brightly? Yes - Next

No - battery

2. Define some attributes of your system. Define 1 or more test cases based on those attributes.

Reliability

- use statistical record keeping

3. Define 1 or more test cases that do "boundary value" testing.

N/A

4. Define 1 or more test cases that "stress" test the system.

Test for cases with more than 1 problem.

5. Define the external interfaces to your system. Define 1 or more test cases to test those interfaces.

User.

6. Define 1 or more test cases to test the system's performance.

Time from when final input is received to when output is displayed.

7. For each question, indicate how the results of each test case will be analyzed (i.e., how you will know the answer is correct).

Use expert for comparison or simulated failures (where we input failure to see if system can detect it).

8. Did the problem description provide enough detail to adequately perform the tests from questions 1-6?

Yes.

9. Develop a "certification" test for your system.

Simulate all possible failures and compare with expert's response.

10. Identify system "disasters" (i.e., things that should not happen). Explain how you will test your system for these "disasters".

- Distributor shaft
- Fuel pump fire

Test these two conditions and insure adequate warnings are given to user when testing.

11. Will your project need the aid of an expert (provide rationale)? If so, indicate the kind of expert required and the type of analysis to be performed.

Yes. Mechanic needs to verify system's logic.

12. Define 1 or more models to aid in your understanding of the system. Document each model.

- Model user interface.
- Cause - Effect diagram.

Handout #9: Exercises on Unit/Integration Test Techniques

1. Pick an implementation approach for your problem. Based on this choice, would you use:

   - Coverage techniques
   - Interprocedural data-flow analysis

Rule-based system.

Coverage techniques:

- Path coverage (cover all possible outcomes)
- Structural (verify each rule fires)

2. Identify "part" of the system that may impact reliability (HINT: you may have to define what reliability is). Define 1 or more test cases to test those "parts".

External inputs.

TC - Simulated inputs

3. Document 1 or more expected sequences of actions for your system.

1st: Gas / Battery

2nd: Starter / Solenoid

3rd: Spark plug / distributor

4th: Carburator / Fuel Pump

4. Is "prototype evaluation" appropriate for your problem? What about mutation testing? Provide rationale.

Prototype the user interface.

No mutation.

5. Exchange your work with another team. Study the problem. Ask yourself the following:

- Does their implementation match the problem?
- Are there any "holes" or inconsistencies in their descriptions?
- Did they pick the right techniques for their implementation approach?

Handout #10: Exercises on Static Test Techniques

1. Identify and define at least 1 "object" in your system (remember, objects consist of both data and operations on that data).

Ignition system consists of [illegible]

2. Write a pre-condition and a post-condition for each operation on the object.

Pre-condition : Gas tank / battery and starter/solenoid must be "good"

Post-condition : Engine [illegible] system

ORIGINAL PAGE IS
OF POOR QUALITY

3. Describe any general properties your "object" must satisfy. Discuss how you would analyze your "object"'s implementation to "prove" those properties are always satisfied.

- Determining failed component.
- Analyze with simulated inputs.

4. Pick at least one operation and defined some rules that implement its specificiation. Ignition system

(defrule spark-plug
F1 (engine turns-over)
F2 (engine dies)
F3 (distributor sparks)
=>
(problem spark-plug)
(conclusion reached))

R1

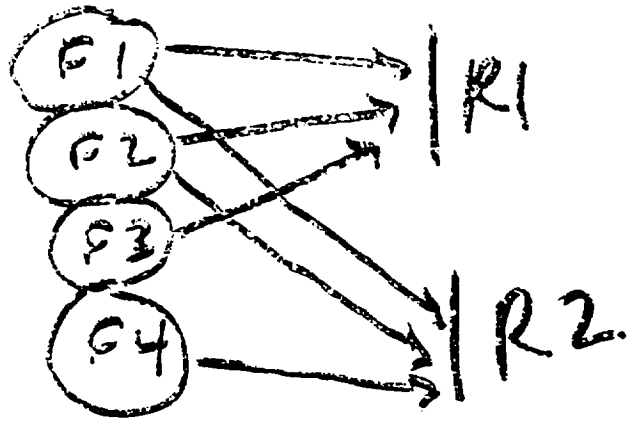
(defrule distributo
(engine turns-ove
(engine dies)
F4 (distributor no-sp
=>
(problem distrib
(conclusion reach

R2.

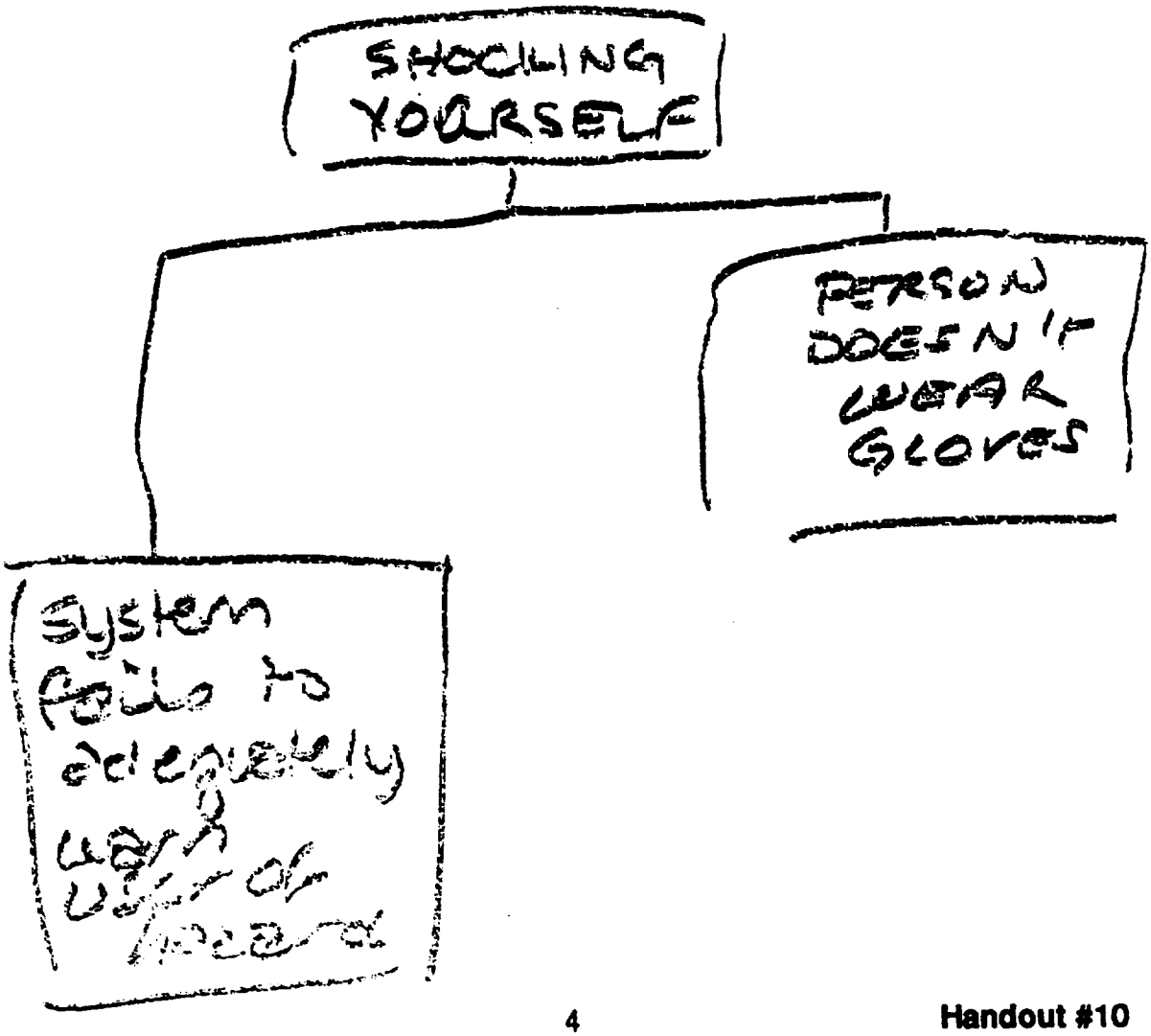
ORIGINAL PAGE IS
OF POOR QUALITY

5. Select one of the following techniques for analyzing these rules. Explain your answer.

- Petri Nets
- Directed Graphs
- Connectivity Matrices



6. Identify 1 "hazard" in your system. Build a fault tree for that "hazard".



7. Idenfity 1 "fault" in your system. Build a fault tree for that "fault".

Handout #11: Exercises on Guidelines

1. Determine whether the recommended approach fits your problem. Identify additional issues that need to be considered.

YES

2. Generate a detailed development plan for your problem. Try to include specific milestones and how they will be achieved.

GATHER REQUIREMENTS
AND EXPERT KNOWLEDGE
- REQ. REV.
PUT DESIGN TOGETHER
- DES. REV.
WRITE CODE
- CODE REV.
UNIT/INTEG. TEST
IV&V
SAC

3. Define specific development increments. Update your plan to reflect those increments.

do user interfaces
last
do each part of the
code and test
after each is complete

4. Consider the test cases you have selected so far. Are there any other kinds of testing you need to do? When will you know when to stop testing?

when all problems
have been proved
to be fixed

5. Build a high-level requirements outline for your system. How well does the original problem definition map to your outline?

FIND THE MOST LIKELY PART TO HAVE FAILED

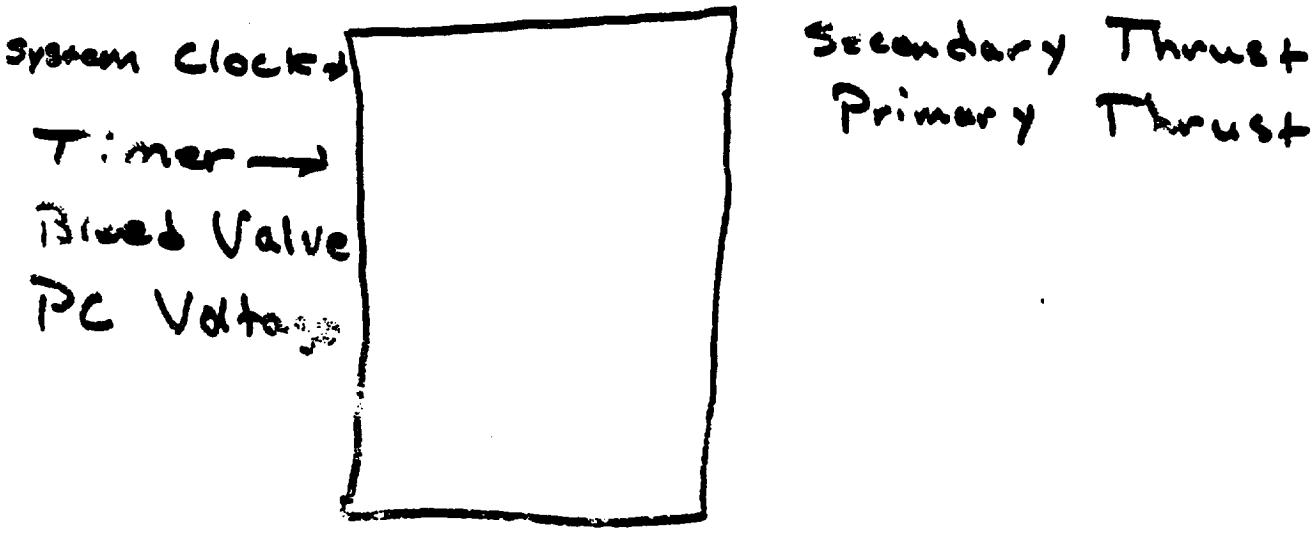
CHOICES:

BATTERY
STARTER MOTOR
STARTER SOLENOID
SPARK PLUGS
DISTRIBUTOR
CARB.
GAS TANK
FUEL PUMP

Team #4

Handout #7: Exercises on General Techniques

1. Define the "black box" view for your system.



2. Identify key terms from the problem description.

Nominal launch sequence functions:
Main Engine Ignition
Secondary Engine Ignition
Terminate direct ground link

Monitor error conditions:
Engine Communication Failure
Engine Failure
PIC ignition Voltage

3. Which of the following techniques would you use? Explain your answer.

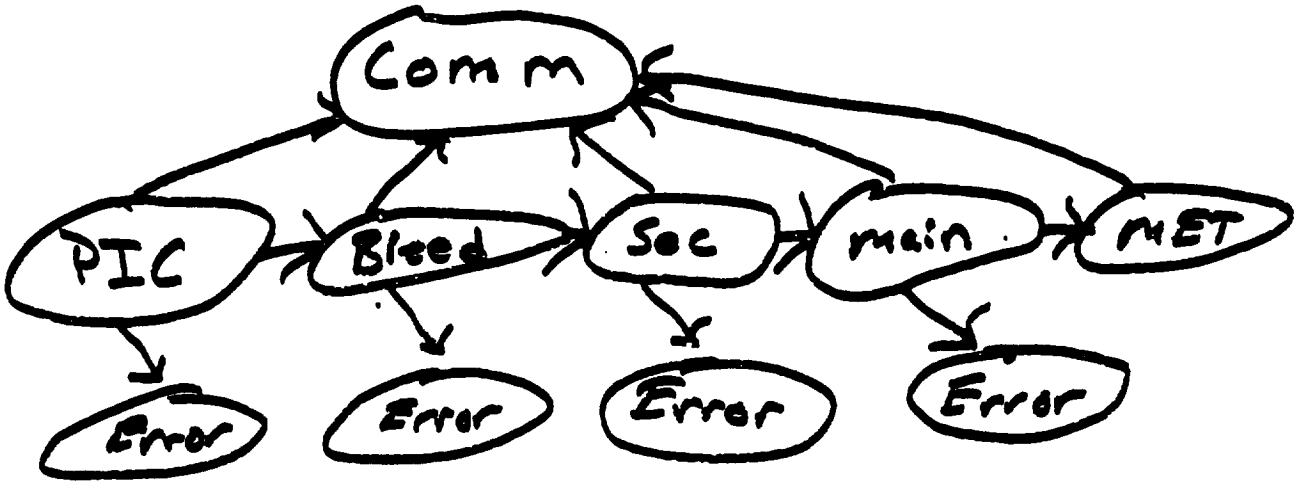
- Prototyping
- Competing Designs
- Independent V&V ✓
- Inspections ✓

Because there is strong detail for nominal cases.

ORIGINAL PAGE IS
OF POOR QUALITY

4. Do a very high level specification for your system using one of the following techniques:

- Decision Table
- Cause-Effect Graph
- State Diagram



Handout #8: Exercises on System Test Techniques

1. Define 1 or more "realistic" test cases for your team exercise.

Constrain Secondary Thrust to 85%.

This should abort.

2. **Define some attributes of your system. Define 1 or more test cases based on those attributes.**

Criticality:

Comm word bit not reset at T+1 and T+2

Should generate Hold message

3. Define 1 or more test cases that do "boundary value" testing.

Structure Data to launch within 2.5 intervals at 4.0 intervals.

4. Define 1 or more test cases that "stress" test the system.

110% thrust on Secondary at T+0.5

5. Define the external interfaces to your system. Define 1 or more test cases to test those interfaces.

Comm Word is external Output all events should effect Comm Word.

6. Define 1 or more test cases to test the system's performance.

Initializing event (push button should result in Launch if no error conditions abort it.

7. For each question, indicate how the results of each test case will be analyzed (i.e., how you will know the answer is correct).

Cases are very specific, Can use pass / fail.

8. Did the problem description provide enough detail to adequately perform the tests from questions 1-6?

Yes but current system description is not "terminal." Not all possible outcomes are defined (i.e. overthrn

9. Develop a "certification" test for your system.

See answers 1, 2, and 6.
Test nominal scenario.

10. Identify system "disasters" (i.e., things that should not happen). Explain how you will test your system for these "disasters".

Main should never Proceed
Secondary. Secondary should
never exceed 100% thrust

11. Will your project need the aid of an expert (provide rationale)? If so, indicate the kind of expert required and the type of analysis to be performed.

Yes, because boundary Valu
are :11 defined and assumpt
need validation.

12. Define 1 or more models to aid in your understanding of the system. Document each model.

Build Computer Simulation.

Handout #9: Exercises on Unit/Integration Test Techniques

1. Pick an implementation approach for your problem. Based on this choice, would you use:

   - Coverage techniques
   - Interprocedural data-flow analysis

Expert System (less code)
Combination because this
is safety critical.

2. Identify "part" of the system that may impact reliability (HINT: you may have to define what reliability is). Define 1 or more test cases to test those "parts".

Clock
Timer
Comm Word

3. Document 1 or more expected sequences of actions for your system.

Nominal → launch

4. Is "prototype evaluation" appropriate for your problem? What about mutation testing? Provide rationale.

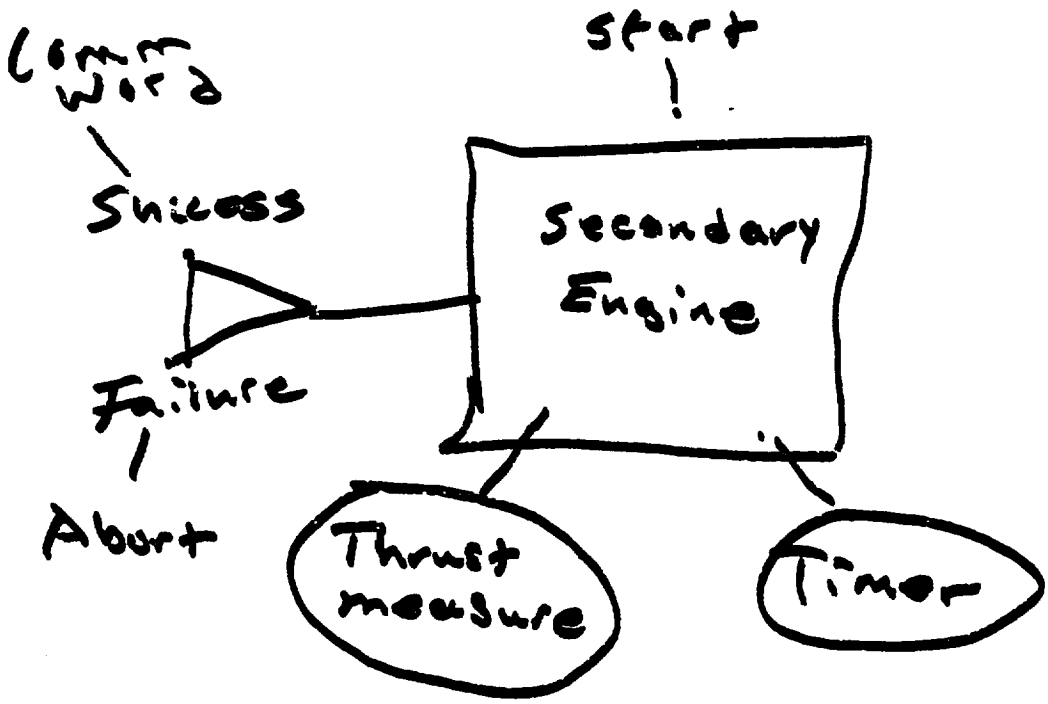
Prototype testing is poss.'ble because of detail describin. nominal versus error condit

5. Exchange your work with another team. Study the problem. Ask yourself the following:

   - Does their implementation match the problem?
   - Are there any "holes" or inconsistencies in their descriptions?
   - Did they pick the right techniques for their implementation approach?

Handout #10: Exercises on Static Test Techniques

1. Identify and define at least 1 "object" in your system (remember, objects consist of both data and operations on that data).



If (Secondary ignition) and
(Thrust ≥ 90%) and
(Timer ≤ 2)
Then (start main Engine)

If (Secondary ignition) and
(Thrust < 90%) and
(Time > 2)
Then (Abort)

2. Write a pre-condition and a post-condition for each operation on the object.

Pre-condition

Propellent bleed Valve (closed)
and ≤ 2 sec from initiate

Post-condition

main engine ready to fire

3. Describe any general properties your "object" must satisfy. Discuss how you would analyze your "object"'s implementation to "prove" those properties are always satisfied.

Tight constraints on time and minimum Thrust.

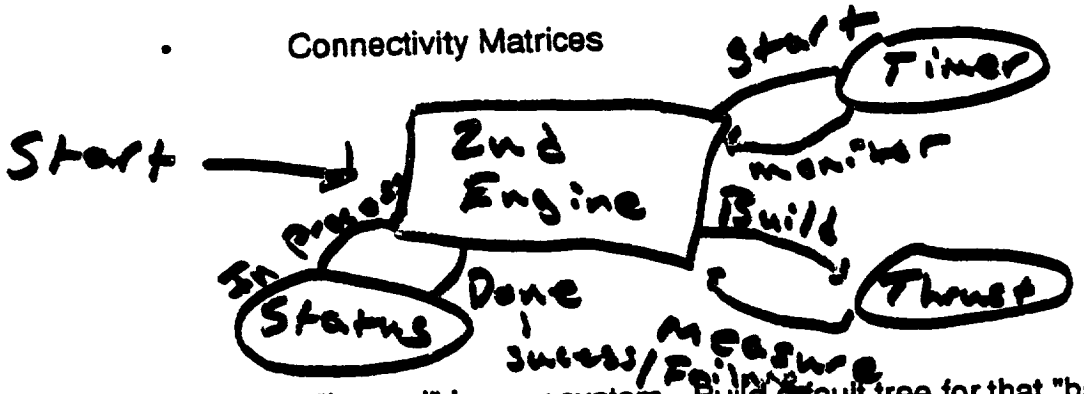
boundary value testing.

4. Pick at least one operation and defined some rules that implement its specificiation.

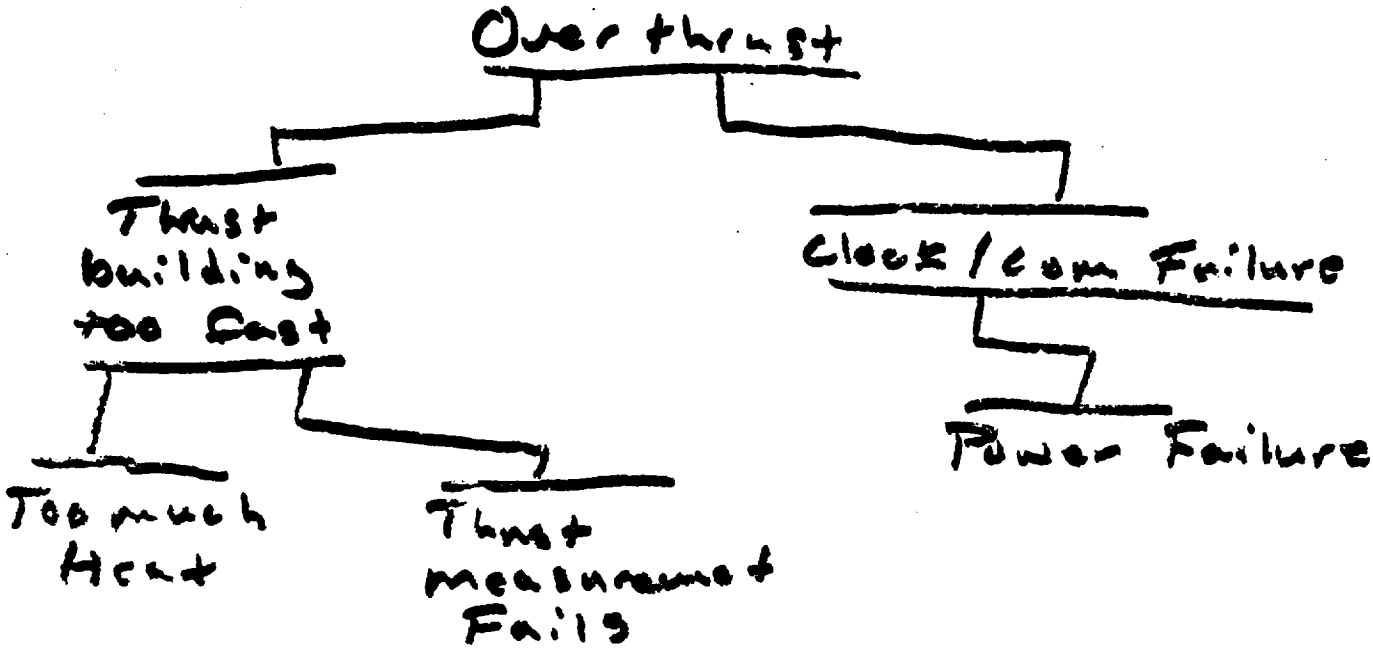
```
If (Secondary Ignition)
    If (Thrust < 90)
        If (Time > 2) then
            (Abort)
    Else If (Thrust ≥ 90%)
        If (Time > 2) then
            (Abort)
        Else
            (Success)
```

5. Select one of the following techniques for analyzing these rules. Explain your answer.

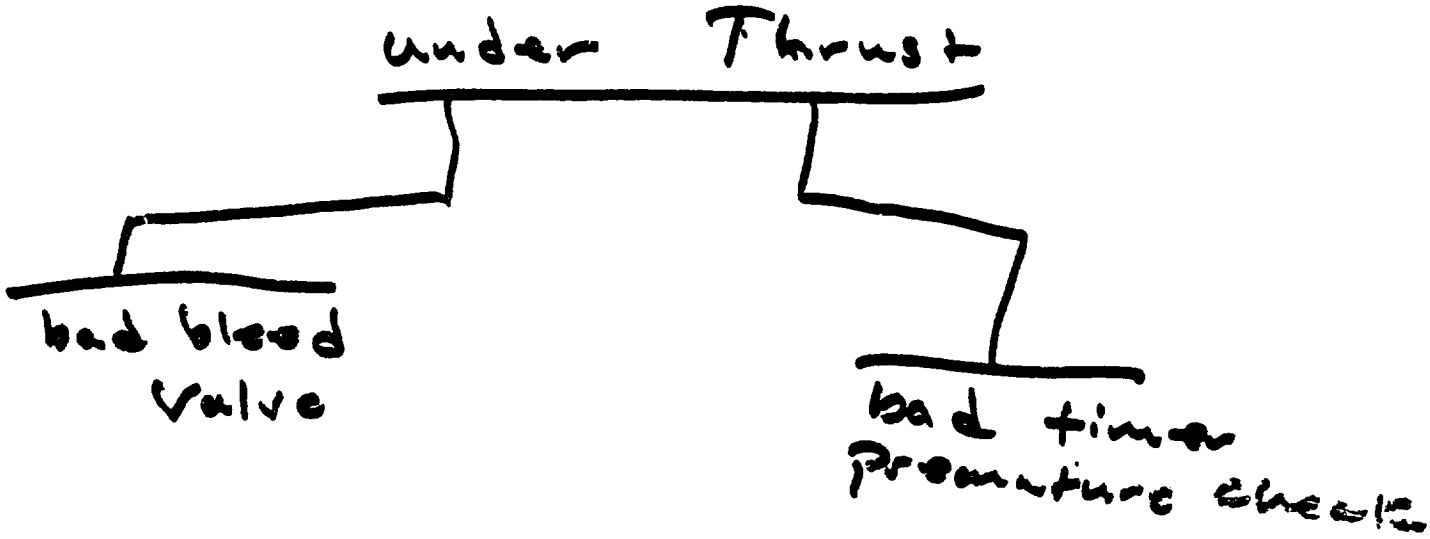
- Petri Nets
- Directed Graphs
- Connectivity Matrices



6. Identify 1 "hazard" in your system. Build a fault tree for that "hazard".



7. Idenfity 1 "fault" in your system. Build a fault tree for that "fault".



Handout #11: Exercises on Guidelines

1. Determine whether the recommended approach fits your problem. Identify additional issues that need to be considered.

Approach is alright. Initial analysis indicates constrains, rules, and facts are incomplet ie overthrust, over voltage, non-adjustable measurement Sampling rates.

2. Generate a detailed development plan for your problem. Try to include specific milestones and how they will be achieved.

modular approach, top down

expert clarification

final requirements

design / code / unit test

integration Test

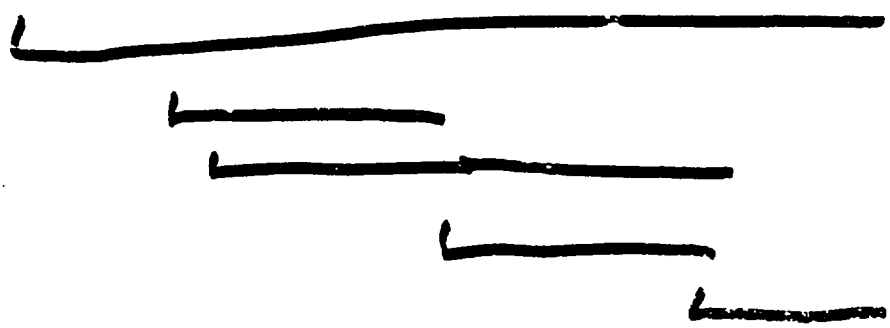
validation Test (w Expert)

delivery

3. Define specific development increments. Update your plan to reflect those increments.

detailed development, assumes final requirements are baselined.

Comm Word
bleed valve/Pic
Timer
Secondary
main



code/test are included for each item

4. Consider the test cases you have selected so far. Are there any other kinds of testing you need to do? When will you know when to stop testing?

Most of our test cases are exhaustive boundary value tests All combinations that should lead to Shutdown must be tes and verified to lead to Shutdown, There must be Tests for normal functions.

5. Build a high-level requirements outline for your system. How well does the original problem definition map to your outline?

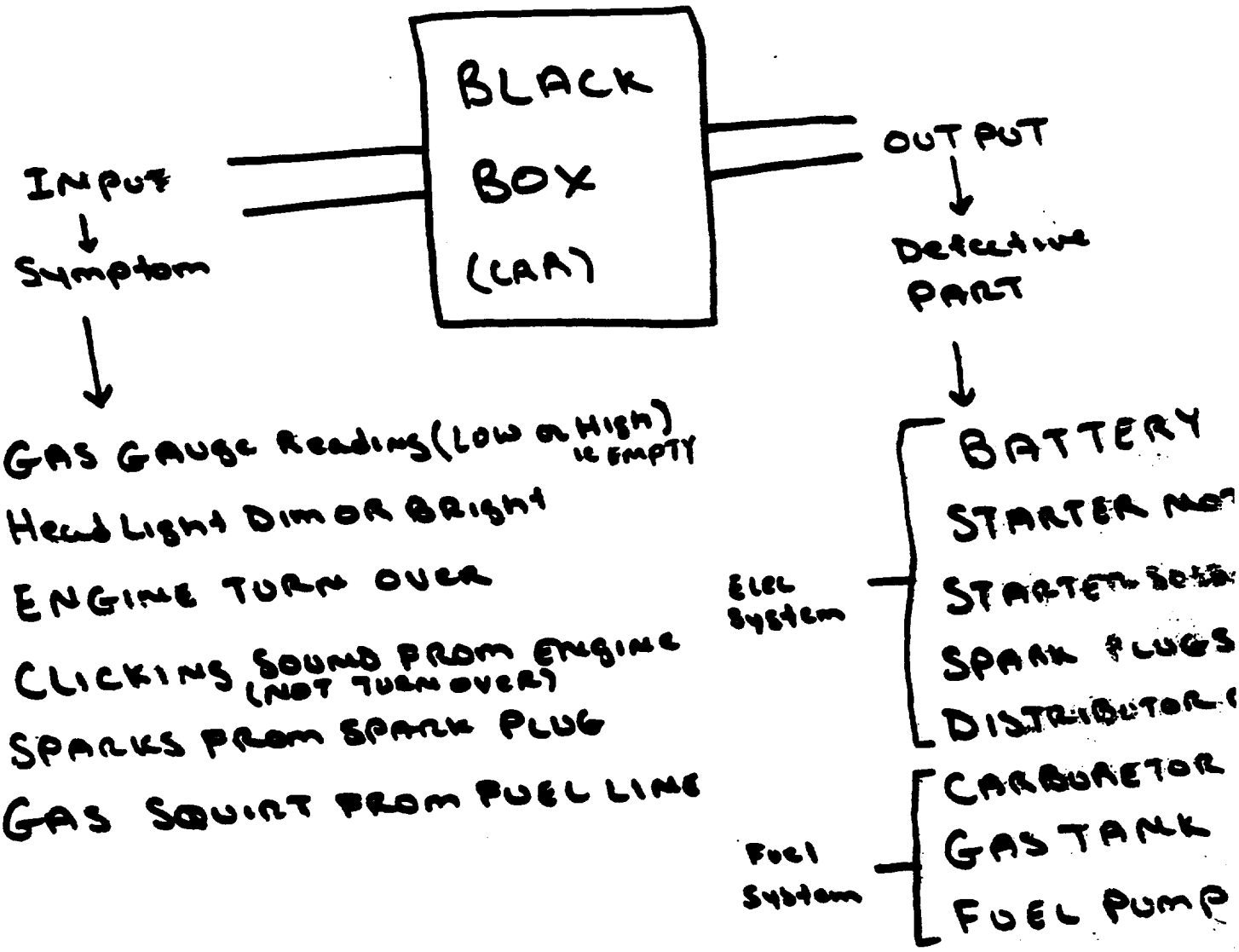
All activities must be controlle by a main scheduling activity. That is, they are started b, acknowledge start, terminate, and report final status to th scheduling activity.

The scheduling activity will use high order bits on the comm status word to acknowledge / not-acknowledge activity start.

All status communications will update a permanent record so that the sequence of activities and their status can be verif after the fact.

CAR WON'T START

Relevant PARTS OF THE BLACK BOX



CAUSE-EFFECT CHART

GAS GAUGE EMPTY
ENGINE TURNS OVER
AND
NOT
CON3
CON4
① GAS TANK EMPTY NOT CON2

HEADLIGHTS DON'T SHINE BRIGHTLY
CON3 (ENGINE DOES NOT TURN OVER)
AND
② BATTERY DEAD NOT

CON2 (GAS TANK NOT EMPTY)
CON2 (BATTERY NOT DEAD)
AND
CON3 (ENGINE DOES NOT TURN OVER)
ENGINE MAKES CLICKING SOUND
AND
AND
③ STARTER

OR

ENGINE DOES NOT MAKE CLICKING SOUND → ④ SOLE

CON3 (ENGINE DOES NOT TURN OVER)
POLL PLUG
AND
AND
⑤ SPARK PLU

SEE SPARKS
OR
DO NOT SEE SPARKS → ⑥ DISTRIBUTOR

CON4 (ENGINE TURNS OVER)
ENGINE RUNS A WHILE
AND
AND
⑦ CARBORETOR

FUEL SQUIRTS FROM THE FUEL LINE WHEN REMOVED FROM THE FUEL PUMP
OR
NO FUEL SQUIRTS → ⑧ FUEL PUMP

CAR WON'T START

1. ANALYZE PROBLEM

- AREAS REQUIRING PROTOTYPING
  - PROCESS INVOLVING ENGINE NOT STARTING
    - STARTER/SOLENOID
    - IGNITION SYSTEM
  - CHECKING THE ELECTRICAL SY.
  - CHECKING THE FUEL SYSTEM.

- HIGH CRITICALITY AREAS
  - ELEC. SYS. CHECK
  - FUEL SYS. CHECK

- EXPERTISE
  - EXPERT MECHANIC FOR KA
  - SYSTEM INITIALLY RELIES ON EXPERT'S KNOWLEDGE, THEN REQUIRES SOME ANAL. BY USER FOR SUBSEQUENT INPU

- SCENARIOS
  - STATISTICAL VALUES WRT PART
  - TEST CASES EXERCISING ALL CAUSES AND PARTS
  - BOUNDRY VALUE TEST CASES
  - MULTIPLE SYMPTOMS

- ES ASPECTS

2. INITIAL PLANNING

- INITIAL SYSTEM BASED ON KA
- NEXT ITERATION WILL
  - UPDATE KNOWLEDGE
  - DEVELOP INTERFACES
  - MUTATION TESTING
- CRITICALITY
  - INITIALLY SMALL
  - W/ SYS. CHECKING - GREATER
- SYSTEM SIZE
  - SMALL BECAUSE OF # OF OPTIONS (PARTS)
  - PRIMARILY PATTERN MATCHI
  - V&V INCORP. INTO DEVELOP.
- MILESTONES
  - MODULE DEVELOPMENT
  - MODULE TESTING
  - UI REVISION
  - UPDATE SYS. AS NECESSARY
- RESOURCES
  - EXPERT (BEGIN & END)
  - DEVELOPER (THROUGHOUT)
  - USER (BEGIN & END)

3. DESIGN + SPEC. ANAL.

- EACH MODULE WILL BE TESTED AS DEVELOPED AND THEN WHEN INTEGRATED

- UTILIZE VARIABLE/FUNCTION LABELS COMMON TO BOTH THE CODE + DOCS.

- EXPERTS INVOLVED W/ QA OF INFO. INITIALLY + DURIN SYSTEM UPDATES.

- STATIC TESTING

  CONNECTIVITY MATRIX

4. INCREMENTS

- SCREENS / WINDOWS

  EXPERT FOR INFO
  USER FOR LOOK & FEEL

- FILES / FUNCTIONS

  TEST LOGIC & PATHS

- OUTPUT

  EXPERT'S VERIFICATION

(1)

Launch Sequencing

Analyze Problem

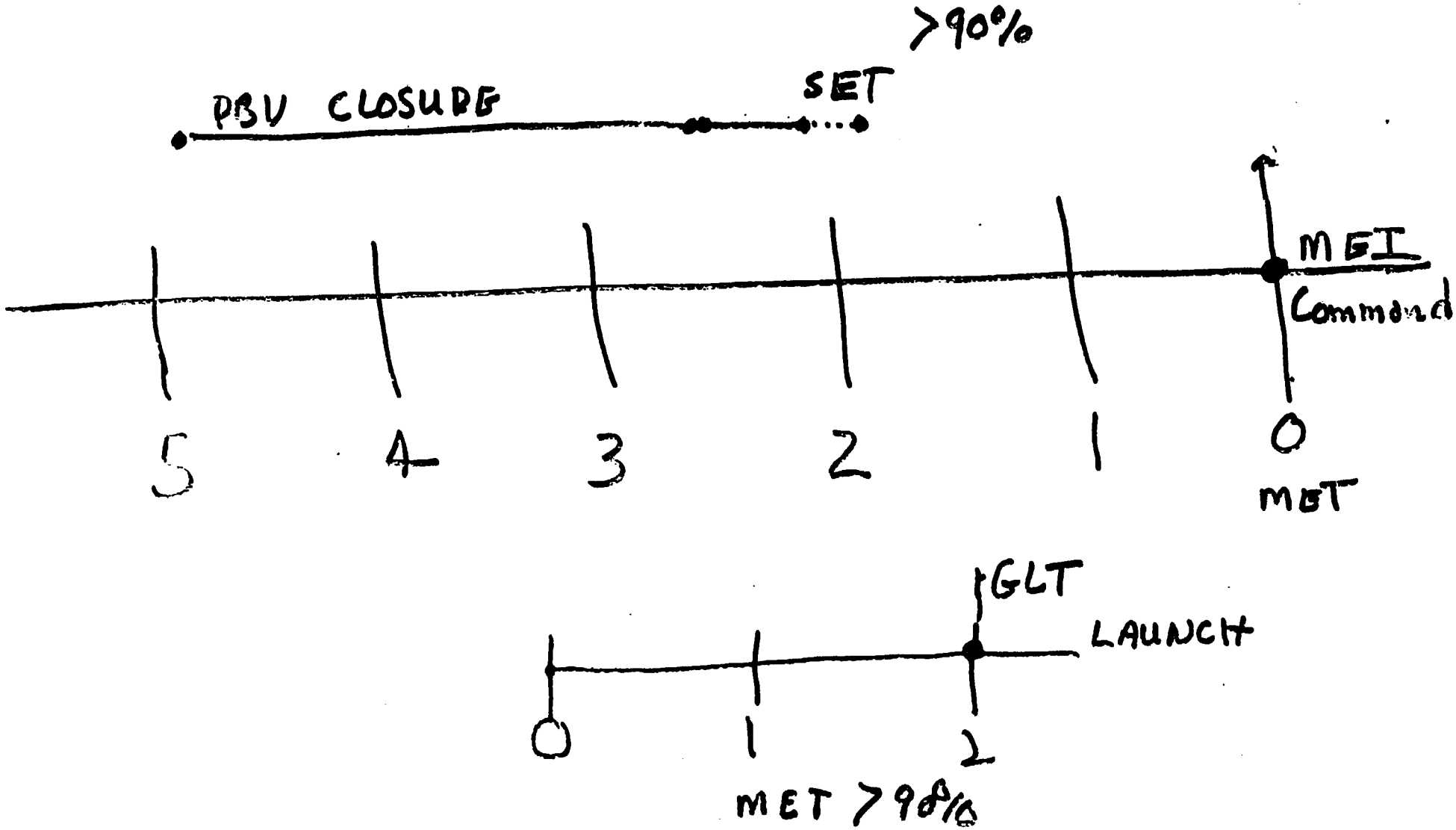
Complexity occur in Main Engine Ignit
Commmd. This depends on secondary
engine thrust build up and is time cri

Communication with ground to monit
critical parameters.

The problem is to be solved by analys
and building a simulation model of th
secondary and main engine interaction.

Operational scenarios follow start
system at a point in time and monit
Engine, valve and thrust parameters
as time progresses.

Testing will be at each critical poin
that an command as issued or thrust
level obtained. Check requirements
to test results.


>90%
SET
PSV CLOSURE
MGI
Command
5
4
3
2
1
0
MET
GLT
LAUNCH
0
1
2
MET >98%

(3)

2. Initial Planning

- Build Engine model - Prototype
  - Fuel system
  - Time system
  - Thrust equations

- Develope simulation of Engine model with ground control

- Update engine model with manufacture specification

- Verify each step with manufacturer specification value

- Milestones are
  - Requirement Anal
  - Preliminary Desi
  - Prototype
  - Critical Desig
  - Model Testing
  - Integrated Test
  - Final Verificatio
  - Ready for Traini

(4)

Perform :

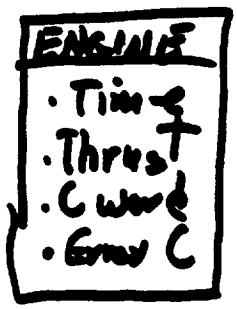
Cause-Effect Graphing

Boundary Testing

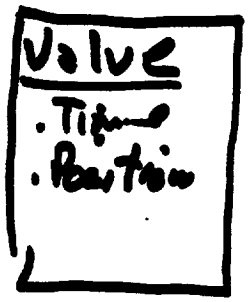
Defect and Disaster Testing

Hazard Analysis

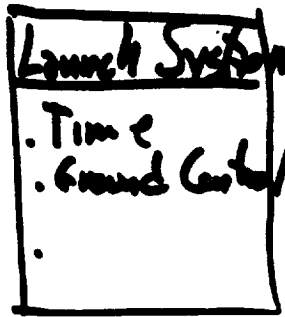
Use Object Oriented Analysis



Operations
- Ignition
- Shut down
- Error Processing



Operation
- Valve Closure
- Propulsant flow

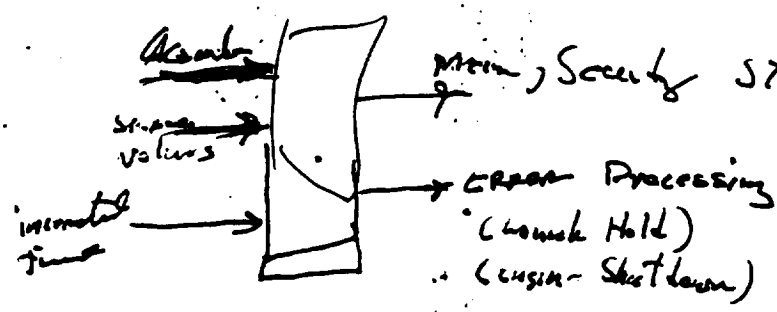


Operation
- Launch Veh
- Error Processing

(1)

Problem:

1) System to Perform the NLSF.
Automated System.



Verify States of Engine Through Time up to Engine Start or simulate

Develop Prototype to Test initial, intermediate, final (each stage)
To ~~costly~~ to use ~~real~~ Flight Hardware
costly

Prototype - full Simulation - Flight Hardware.
Real

Risk - Simulation Data for Flight Hardware
- Must have engine model.
- Concurrency.

---

(2)

Key Terms:
Ignition
Incremental Time
Sensed Values
Error Processing
MET

(3)

Prototyping - Inspection

Prepare a space vehicle for lift off

↓

using on-board functions

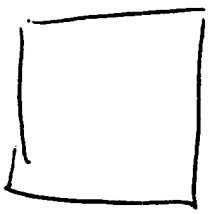
Perform pre-launch activities & check quickly.

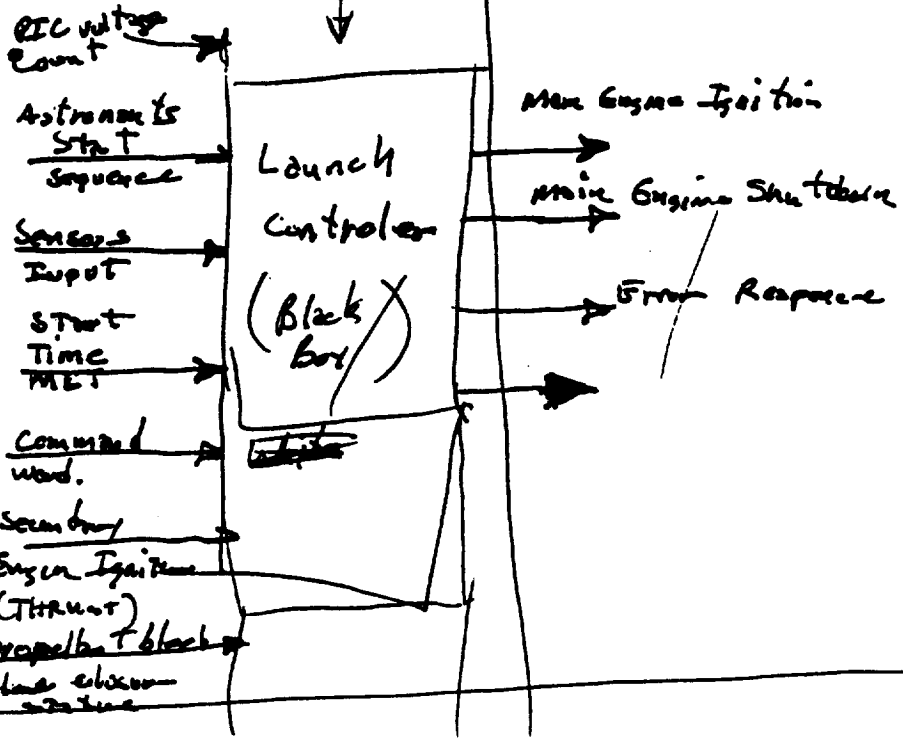
Develop a fast approach

Consider Safety

Consequence of a wrong decision

Resources	implementation	Requirements
• Astronauts - start sequence • Launch Experts - Engineers • Programmers • Schedulers • Ground Communication	• Clock	





(2)

Decision Table
Cause - Effect Graph
State diagram

PIII 8
Minimum Complexity Testing

≠ Simulate to Explicit Modeling

The most important Technique

(2)

Decision Table:
Left Right.
Condition :: action
Row is called rule.

Rules

1) IF MET = 0.0 and secondary engine has Ignition between 2 and 2.3 secs Then Issue Command to "Main Engine Ignition".

2) IF ~~"Propellant bleed valve closure" and Secondary Engine Ignition~~ 2 secs has not occurred and "propellant bleed valve closure has occurred" Then issue "Secondary Engine Ignition"

3) If "Secondary Engine ~~Ignition~~ thrust ≥ 80 % command "Then record "Secondary Engine start" complete

4) IF "main engine ignition complete" Then "Terminate direct ground link"

5) IF "Terminate direct ground link command issued" and "ground link Termination confirmed" Then "Nominal Launch Sequence fi

6) IF command to "Main Engine Ignition" occurred and "MET increment of .1 sec" occurred" and "Engine Command Word BIT 1 not reset" Then note "Engine command failure".

(5) Exercise IU-12 — Launch Sequencing

1) Implementation approach

- Use ADA

Use Condition Coverage. The Launch Sequ... problems involves function occurring in prop... order and Testing parameters in order to dri... The next step.

2) Parts of the System. elements of

The system consists of Valves, Two types of... Engines and communication with ground. Al... elements must work reliably (no failures) ... for the total system to work (Engine Ignitio... Each part will be tested separately (Subsy... is integrated together (System). Order is import...

Subsystem = System + Order testing.
(Sequence)

3) Valve open
Secondary Engine Ignition
Thrust & Time build up
Main Engine Ignition Thrust build up
Ground Control Termination

4) Prototyping is the way to go (Can't destroy ... real Engine by testing). No Mystatic. Test ... criteria with manufacture specification.

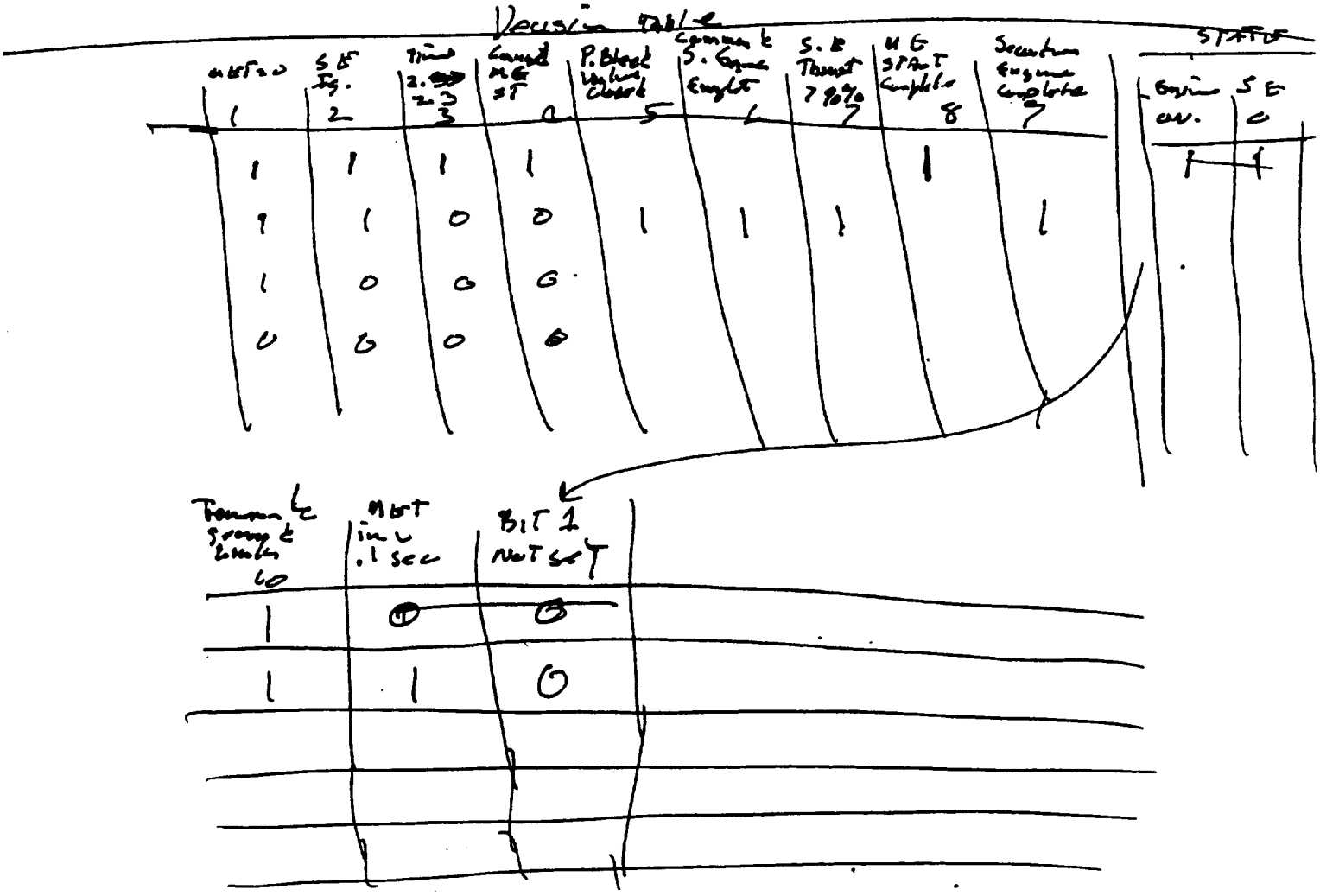
(Q3)

7 IF "Engine Thrust Lower Than 90%" and "Engine Command Issued in This 2sec" note on monitor "Engine Failure".

~~If "Engine Communication Failure" occured~~

9 IF "Engine Bit 1 not reset" and "Two consecutive Engine Shut Commands issued" then "Implement Engine Communication Failure Recovery t

9 IF "In Engine Communication Failure mode" perform The following -



(4) 1. Realistic Test Cases (103)

- Secondary Engine Ignition within 2 sec of propellant bleed valve closure
  Clock failed - no start
  Check bleed valve open + close
- Main Engine Ignite within time limit of Secondary Engine

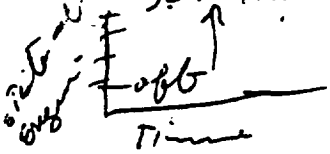
2) Reliability of HW - always work → main engine b.T flip flop.
   Position of bleed valve
   Time Indicator - very critical (No failures in Clocks)

3) ET exit, Top Time boundary, the Engine Ignite
   Monitor indication of termination of ground link at engine ignition
   no fault left.

4) continuous flip flop (open/close) after
   bleed valve
   Engine perform up to 90% in .5 sec
   (continuation of Thrust to Thrust)

5) Time pulse
   Ground Control
   Observer - Monitor Control → Fuel System

6) Engine ignition with limits and reach Thrust of 50% in 5 sec

7) Graphics
   Secondary Engine
   Main Engine
   

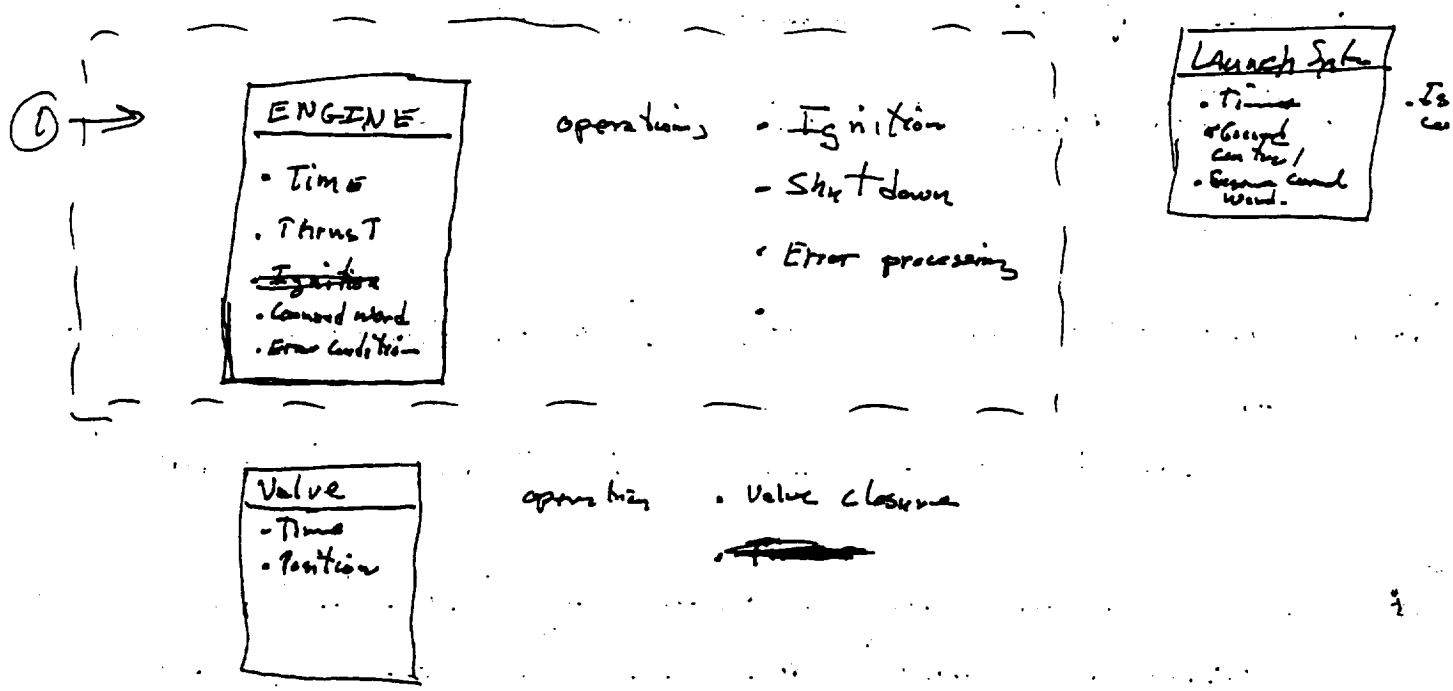
8) No - Need more White Box parameters

9) Engine perform within Engine Specification to
   publish reliability.

10) Disturbance 70% Thrust to force to restart - Separation
    Force Limits 70% → Shut down + get behind
    explosion wall.
    (only simulate ~~disturbance~~ disaster)

11) - NO : Highly HW intensive, Need Manufacturing Report
    interpret results.

6/27/92 STATIC Execution



(2)

Pre Condition : ~~Not Ign~~ OFF

Ignition

Post Condition : 90% Thrust
Ignition, ~~Ground lock Pneumatic~~

Pre Condition : Engine Shut

Shutdown

Post Condition : ~~Engine Command Word Not Safe~~ = No Thrust

Pre Condition : No Error - Engine Command Word OK

Error Processing

Post Condition : Launch hold, Engine Shutdown.

(3) · Command word B,T 1 sat

· Thrust ≥ 90% 2 seconds after start

· Thrust building to 90% during 2 seconds

(4) Ignition

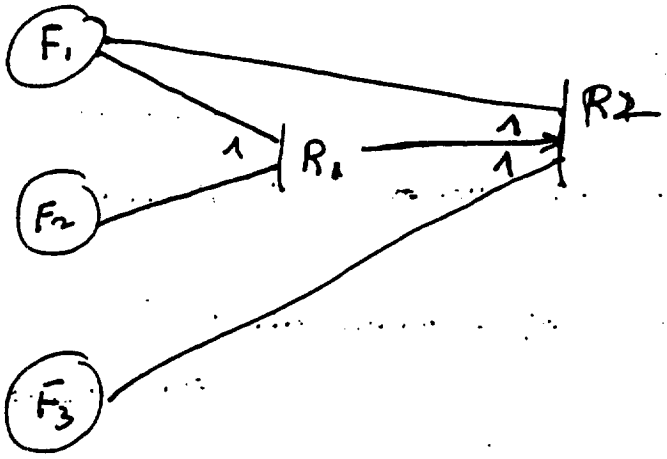
Rule (2) IF MET = 0.0 and secondary engine has Ignition between 2 seconds

Then: Issue Command to "Main Engine Ignition"

Rule (3) IF 2 seconds has not occurred and "propellent bleed valve closure has occurred" Then issue "Secondary Engine Ignition"

(5) Petri Net

Fact		Rules
F_1	Time is 2 sec	Rule (1) = Time build up 2 seconds
F_2	Propellant bleed valve close	2 = if propellant valve closure occurs - Secondary Eng
F_3	MET = 0.0	



(6) Hazard - Clock failure, no Ti
How: Battery failure. No
Verify · Check all electric connec
Verify Clock works be
Launch Sequence start.

• ~~Uncertainty~~ Requirements

Launch Sequence MET = 0.0

Build up Second Engine Thrust 80%
within 2-2.3 seconds.

• ~~Secondary Engine~~ Main Engine Ignition
Terminate Guarantee link.

Blood Valve Closure - within 2 seconds
Engine Thrust > 80% in .5

Secondary Engine Ignition

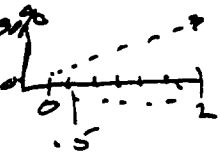
Monitor

Engine Command ~~Word~~ Bit

Time .1 sec.

~~Failure Confirm~~

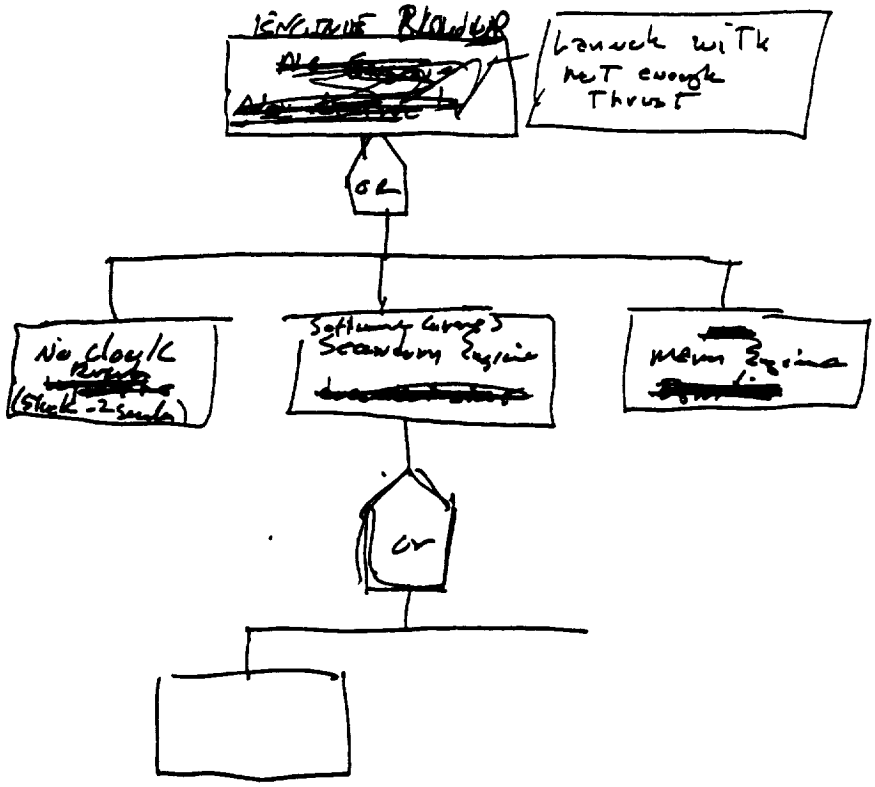
~~Failure~~ Primer Light ~~Bit~~



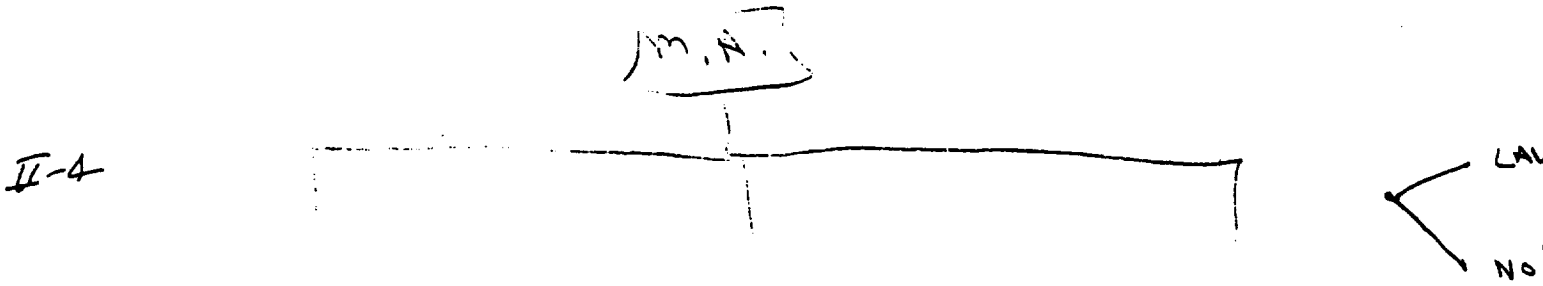
Thrust < 80% at time = 2 sec. =
Secondary Engine failure

~~AOR~~ ~~R&T~~

Fault Tree for Clock failure



Blood Valve stuck Closed



. Main Goal is to get Main Engine Ignition

Requirements

1 Propellant bleed valve shall release fuel into Secondary E.
2 within Two seconds after Propellant bleed valve closure the Secondary Engine ~~will~~ shall have Ignition.
3 Secondary Engine shall obtain >90% thrust
4 After Secondary Engine thrust build up to >90%
5 * Main Engine Ignition command shall be issued after Secondary Engine thrust build up to >90%
6 Main Engine Command shall be issued between 2 and 2. seconds after secondary Engine thrust has built up to >90%
7 Ground link shall maintain contact with vehicle until main engine ignition
8 Ground link shall Terminate at main engine ignition
9 Ground link shall monitor each 1 second the Engine Command Word Bit
10 Engine Command Word Bit 1 shall reset when Main engine ignition occur.
11 Ground shall arm RTC ignition voltage and monitor for.
12 Engine Communication Failure shall occure if Engine Command Word Bit 1 not reset on two consecutive commands.

The Problem:

As initially stated is to feed a banana to a monkey. The information supplied by the customer suggested a much too complex system but discussion showed that the real requirement is to measure the relative intelligence of a monkey. The system constructed from the requirements first submitted was invalid provided that some performance time measurements were added.

Life Cycle Model

The Life Cycle model chosen for the development was the European Space Agency Model.

```
IF
((Is-Red)(Holds-Key)(Is-object-Key)
   ...
!!
Assert ((On-top-of Floor)(ladder)
Assert (On-top of Floor)(Red-Key)
```

Build prototype that assumes cooperative intelligent monkey.

* Note: Will need an agent to motivate the monkey to move.

A prologue trace would provide much the same information as the connectivity matrix in that all paths would be shown over whatever their lengths.

Operation

move Monkey on top of pillar holding red key

Pre Con

On floor at T2-2

Post

* note monkey on top of pillar holding red key

Eff

Monkey unlocks Chest

Des

Monkey on top ~~pillar~~ holding red key
red chest unlocked and ladder on top of red chest
* note No specification for the weight of the ladder.

The monkey drops the ladder on the floor monkey drops red key on
MONKEY jumps on floor
Move monkey from T2-2
Move monkey to ~~blue~~ T8-8 holding ladder
Monkey climbs on blue couch holding ladder.
monkey puts ladder on blue couch. monkey jumps down
~~monkey climbs on ladder to green chest~~ floor.
Monkey moves back to T2-2 obtain red key
Moves back to T8-8
climbs on top of blue couch ~~holding~~ ladder holding
red key.
climbs back up ladder to green chest, opens it and
gets the blue key. drops ladder to floor
Climbs back down to blue couch, jumps to floor
~~picks up blue key moves to drop it~~ picks up the ladder
Moves to T7-7, puts ladder, moves back to T8-8
picks up blue key, returns to T7-7, climbs
ladder, opens blue chest and gets banana.

1. The monkey is the object. The operation is to bring the monkey and banana to the same location.

Operations

1) Move monkey to floor

Pre-condition

Monkey on top of green couch at T5-7

Post-Condition

Monkey on top of floor at T5-7

Move monkey to location T1-3

Pre-condition

Monkey on top of floor at T5-7

Post-condition

Monkey ~~at~~ ~~T5-7~~ on top of floor at T1-3 holding the red key

~~Pre~~ Operation

Move monkey to T2-2

~~Pre~~

Move monkey to big pillow

~~Post~~ Monkey on top of floor at T1-3

Post

Monkey on top of floor T2-2

the operation and its definitions rules that implement its specifications

Attachment C

Name	Org	Phone #
Jerry Klumes	CAE-LINK	280-4559
Paul Sherman	CAE-LINK	280-4560
Danny Strauss	Intermetrics	480-4101
Robert St John	Intermetrics	480-4101
Tom Callaghan	Lockheed	333-7820
Ulysses Hunter	Loral	335-6733
Tuan Pham	Loral	335-6573
Darrell Harlan	Unisys (Paramax)	282-4679
Debra Haddox	Paramax	282-4689
Glenn Lewis	Paramax	282-4672
Kathi Noyes	Paramax	282-4531
William Morris	McDonnel Douglas	283-1244
Patrick Kenney	McDonnell Douglas	283-4332

July Sign-in List

Name	Company	Phone #
Al Brown	Lockheed	333-6853
Ray Runnels	UNISYS/PARAMAX	282-4665
DAN DREW	PARAMAX	282-3664
Ross Pettingar	Lockheed	333-7314
AL CARLTON	CARLTON & ASSOCIATES	280-4351
LESLIE AMBROSE	MITRE	333-[illegible]
Giau Ho	Unisys	483-2941
Russell Sellers	GHG	488-8806
Dave Kenning	GHG	488-8806
Ajmal Tajani	GHG	488-8806
Kim Brandt	Paramax	2822534
Holly Gibbons	Intermetrics	480-4101
Michael Freeman	Loral	335-6775
Tom Thomsen	Loral	335-6714
[illegible]	[illegible]	[illegible]
[illegible]	Paramax	282-2000 x 7[illegible]2
Alice Lee	NASA/JSC	483-5234
BLAKE AFRERE	INTERMETRICS	480-4101
Richard Nguyen	LORAL	335-2529
Ingrid Ku	CSC	280-3654

Sign-In

Name	Company	Phone #
Tod Sandman	Lockheed	333-6712
Nina Tammi	Loral	335-6911
Sunil Fatedar	MDSSC	283-4329 (483-
Ruth Hirt	LORAL	335-6583
William Morris	MDSSC	283-1294
Arland Actkinson	Dual	486-1984
SATYANARAYANAN SESHAN	GTHG	488-8806.

Attendence List

NAME	ORG	PHONE
HUY NGUYEN	IBM	282-8340
CHARLES GREER	IBM	282-7228
TRACI Bardelmann	CSC	280-2335
Stephen Bandendistel	CSC	280-2430
Lewis George Jr.	LORAL	335-6422
GEORGIA WARD	LORAL	335-6507
TOM JONES	LORAL	335-6424
Ken Harris	LORAL	335-6829
Russ Walker	Loral	335-6631
Celia Taylor	LORAL	335-6520
Chris Culbert	NASA/PT4	483-8080
Joe Nieten	GHG	488-8806
Bob McNenny	MDSSC	283-4325
Laurie Howard	NASA/EI223	244-5132
José Escobedo	G.H.G	488-8806
DON CULP	RSOC	483-0891
Sherry Land	JSC/ER2	483-2066
Pratibha Baloor	UHCL	283-37[illegible]
Bob Voss	LORAL	335-65[illegible]
Grace del Puerto	Unisys	282-4657

ESMF workshop Attendance List

David Hamilton	IBM	282-3[illegible]
[illegible]	[illegible]	[illegible]
Tom Sarzo	CSC/PT4	483-9246
Chris Ortiz	NASA/PT4	483-19[illegible]
Bebe Li	NASA/PT4	483-80[illegible]
Luis Walk	"	3-80[illegible]
Brian Donnell	"	380 7[illegible]
Pratibha Baloor	UHCL	283-375
DAN BOWMAN	IBM	282-766
Scott French	IBM	282-

Attachment D

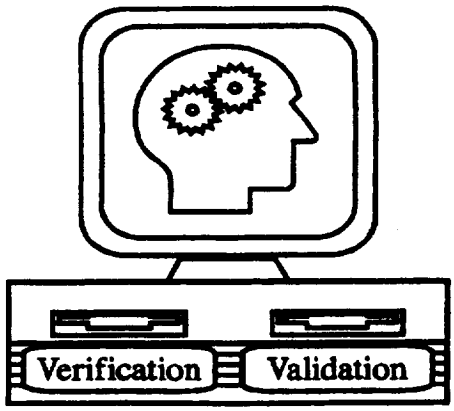
Overview of Verification and Validation of Expert Systems

Authors:

Scott W. French
FRENCHS@HOUVMSCC.VNET.IBM.COM

David Hamilton
HAMILTON@HOUVMSCC.VNET.IBM.COM

IBM Corporation
3700 Bay Area Blvd.
Houston, TX 77058

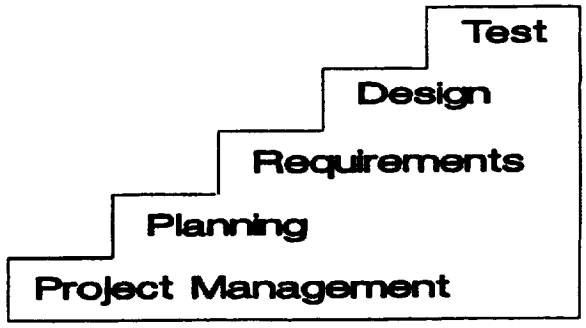


Welcome

Welcome to the Overview on Verification and Validation (V&V) of Expert Systems

This Overview provides

- A view of the "state of the practice" in V&V of Expert Systems
- Insight into what we have taught developers about V&V
  - » Many Techniques
- Guidelines for management's role in V&V
  - » Developers stressed need for management involvement



"State of the Practice" in Expert Systems V&V

Significant research has been done in Expert Systems V&V

- Developed conceptual approaches
- Proposed various techniques

No significant case studies or field demonstrations

Research is based on many conjectures about how Expert Systems are built

- Expert Systems have no requirements
- Small V&V effort for Expert Systems compared to other software
- Testing an Expert System is hard

"State of the Practice" in Expert Systems V&V ...

Survey state-of-the-practice in ES V&V

- Determine real issues in V&V of ES
- Assess accuracy of conjectures
- Impact future work in V&V of ES

60+ projects were asked questions such as[1]

- V&V activities done, not done
- Issues that occur in practice
- Extent to which V&V impacts issues
- User views of quality/reliability

Caveats

- Results are not statistically valid
- Responses reflect opinion

[1]The survey did not attempt to assess whether a given system was good or bad. Our goal was to uncover issues encountered.

"State of the Practice" in Expert Systems V&V ...

Major difficulties developers experience when building Expert Systems

- **Determining when to stop testing (63%)**
- **Validating knowledge acquired from the expert (60%)**
- **Managing the complexity of the problem being solved (40%)**

Process used in building an Expert System

- **22% followed no life-cycle model**
- **43% built *operational* prototypes**
- **14% cited Configuration Management as an issue**
  - » **One-on-one interviews indicated greater concern**

"State of the Practice" in Expert Systems V&V ...

Methods used in verifying and validating an Expert System

- **57% operational systems had no requirements**
- **52% used only one technique**

Resulting quality of the Expert System

- **Considered both developer and user perspective**

Item	Dev	Users
Evaluation is difficult	27%	100%
Less accurate than Expert	44%	80%
Did not meet expectations	49%	100%

Addressing the Issues

Survey indicates ES projects need help

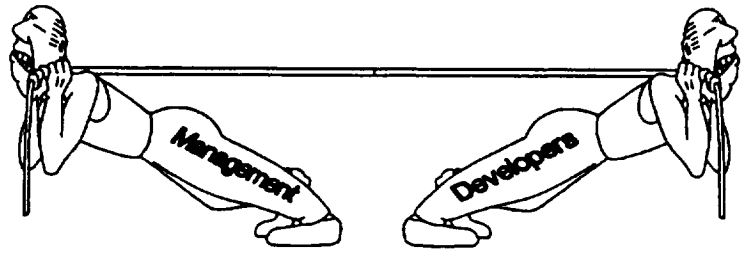
Two presentations address those needs:

- **Management overview of V&V**
- **Developer instruction in doing V&V (Workshop on V&V of ES)**

These presentations seek ...

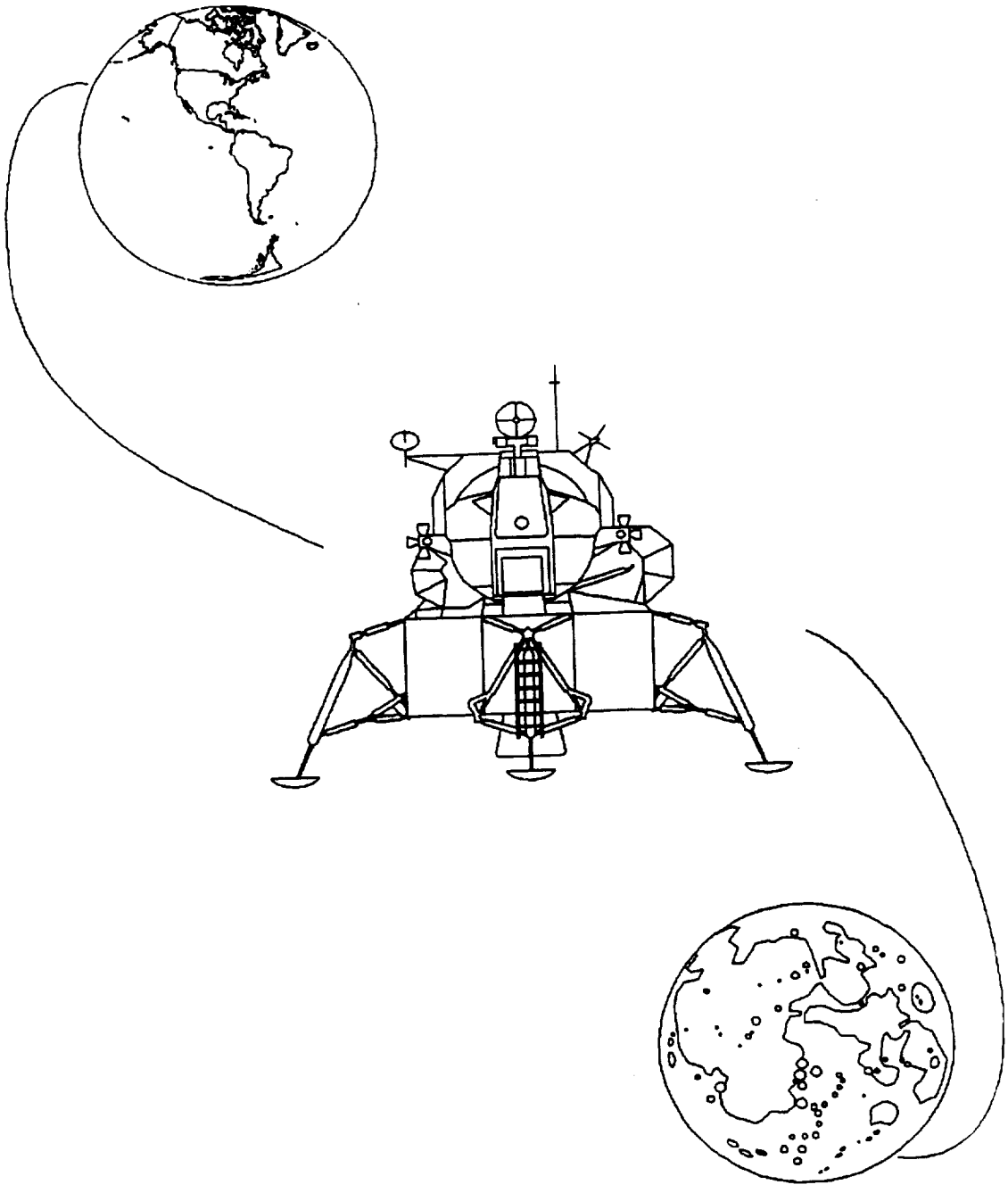
To help project members "pull together" for higher quality results ...

And avoid this!



Additional work still needed in many areas

The Apollo 11 Scenario



ES Developer Workshop on V&V

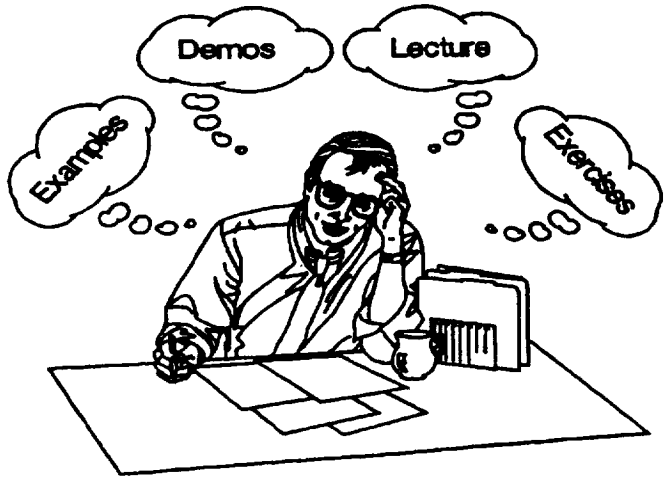
Workshop is taught over a 4 day period

Goal is to help developers do their job better

Many topics covered

- **Theoretical basis for V&V**
- **Planning for V&V**
- **47+ techniques covered**
- **Guidelines for doing V&V**

Developers learn by



Key Points Developers Learn

What is Verification and Validation?

Verification: Am I building the product right

- **Did I do what I was told to do**
- **Most techniques address this**
- **Therefore, generally easier to satisfy**

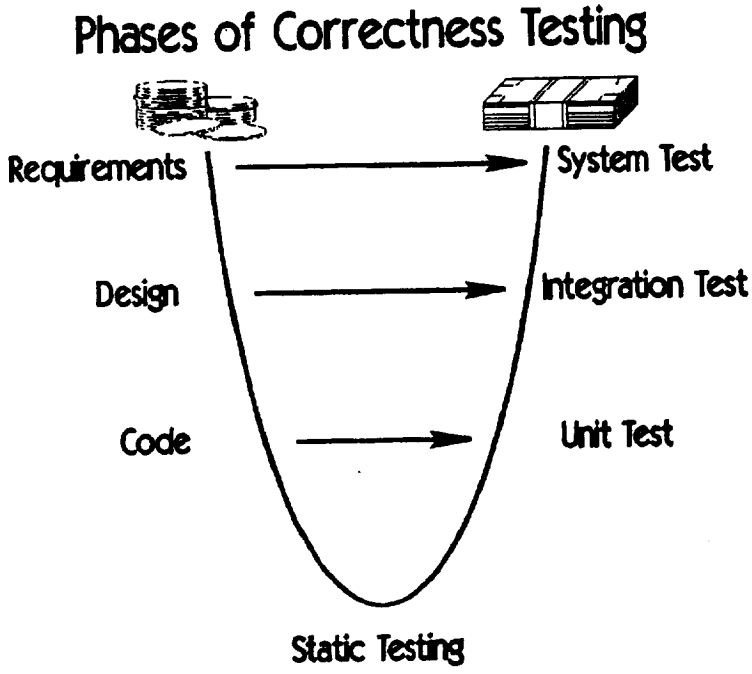
Validation: Am I building the right product

- **Was I told the right thing to do**
- **Few techniques help with this**
- **Therefore, generally more difficult to satisfy**

Key Points Developers Learn ...

Focus on finding errors early

- Follow a "test as you go" approach
- Emphasize human analysis



"It is not uncommon to spend 30 to 50 percent of the ... cost ... for the verification effort using the after-the-fact approach"6

Key Points Developers Learn ...

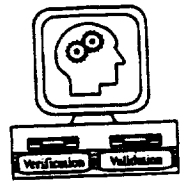
Spend more time analyzing the problem

- A complete understanding of the problem is never initially possible
- Will use prototyping to model their understanding of user needs
- Translation: "Pay me now or pay me later"

Insist on following a development life-cycle

- No more "operational" prototypes

"Building large programs is NOT like building small ones and software engineering is different from most other engineering disciplines"[5]

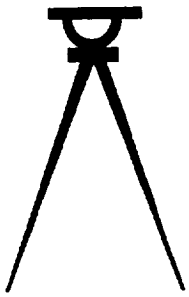


Key Points Developers Learn ...

Plan for V&V

- **Match implementation to problem**
  - » Solution-oriented vs. Technology-oriented

"Many problems that occur ... are the result of ... generating code without thinking about the design."[6]



Survey indicated 45% of Expert Systems mix conventional and procedural code.

- **Identify required resources**
  - » Hardware, Software, expertise, ...
  - » All impact the project's feasability
  - » Should be done early instead of later

Key Points Developers Learn ...

Plan for V&V ...

- **Prioritizing tasks (e.g., do the critical things first)**

"A comprehensive test management approach recognizes the differences in objectives and strategies of different types of testing."[10]

- **Remembering that the system will have to be maintained**

For every dollar spent in development, two dollars is spent on maintenance.[2]

Key Points Developers Learn ...

Build a description of the problem

- Have something to test against

"If an expert system starts with vague objectives, some may conclude that it doesn't matter what the eventual system does, because anything is better than nothing."[4]

- Must be a "crisp" definition

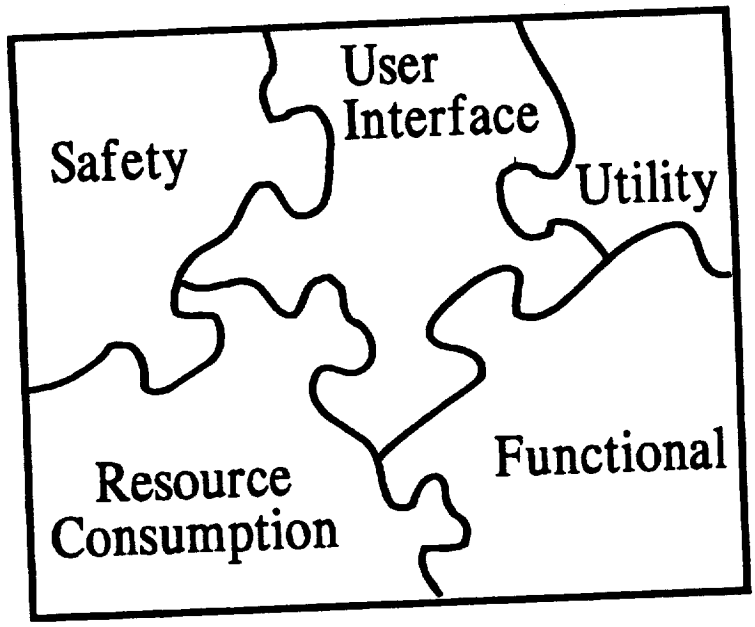
"Knowledge-based systems have a greater liklihood of succeeding - and, in a sense, of being valid - when they address a narrowly defined problem."[7]

Key Points Developers Learn ...

Focus on doing "smarter" testing

- Matching the right technique to the right problem in the right situation

The Verification Puzzle



- Develops an understanding of why the system is correct
- Know when to stop testing

Key Points Developers Learn ...

Expect the system to work

- **Confidence in applying techniques**
- **Confidence that the appropriate issues have been considered**
- **Confidence that the problem can be solved**

"A good programmer understands what his program is supposed to do and why he expects his program to do it."[5]

"The difficulty with low expectations is that they become self-fulfilling."[5]

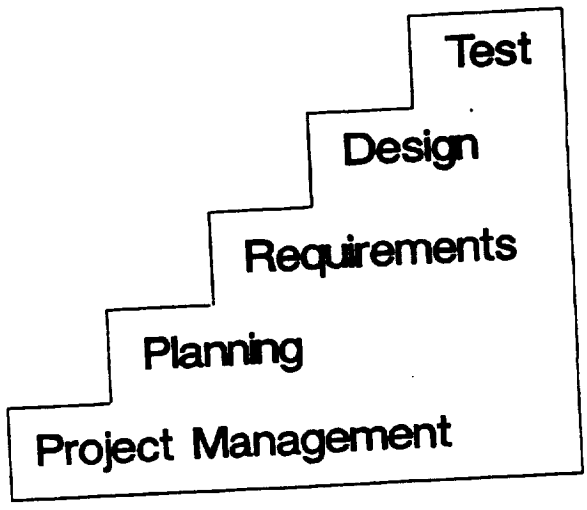
Guidelines

The following are guidelines to be followed when applying V&V to your project

You may want to do these yourself or delegate them to members of the development team

Just make sure they happen

Guidelines apply to the following steps



Guidelines

Project Management

- **Include V&V as part of the cost of developing software**
  - » Spread throughout the development cycle
  - » Not all at the end
- **Allocate resources for V&V**
  - » Be prepared to postpone a project if the resources are not available
  - » For expert systems, you will need the expert's time for V&V
- **Make sure you follow a systematic development approach**
  - » Best bet is to use a life-cycle model that includes major testing phases
  - » Focus on a "test as you go" approach

Guidelines ...

Project Management ...

- Make sure your plan is based on the system's characteristics
  - » What problem is to be solved
  - » Complexity of that problem
  - » Effort required to generate a solution
  - » Types of correctness that matter
- Include prototyping to help validate understanding of the user's needs

"The only question is whether you or your customer will discover them (errors)"[8]

"... there is now less excuse than ever for not involving users early on ..."[1]

  - » Separate prototyping from complete system development

Guidelines ...

Problem Analysis

- Narrow the scope of the problem as much as possible
  - » Better to have a system that solves one problem really well than a system that solves many problems poorly
- Do not force the solution to be an expert system

Requirements

- Write Requirements
  - Impossible to prove anything about the system without it
  - » Consider all kinds of correctness
- Make sure that (at a minimum) the expected use of the system is defined

Guidelines ...

Design

- Pick methods that make static analysis easier
  - » Easier and less costly system test
- Map requirements to design
  - » Helps decide if anything is missing
- Pick a reasonable design notation and stick with it

"... conceptual integrity is the most important consideration in system design. It is better to have a system ... reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas."[3]

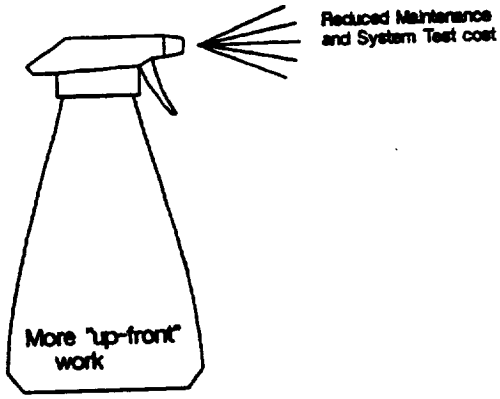
Guidelines ...

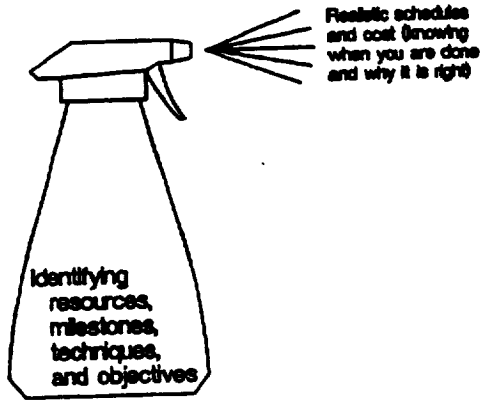
Test

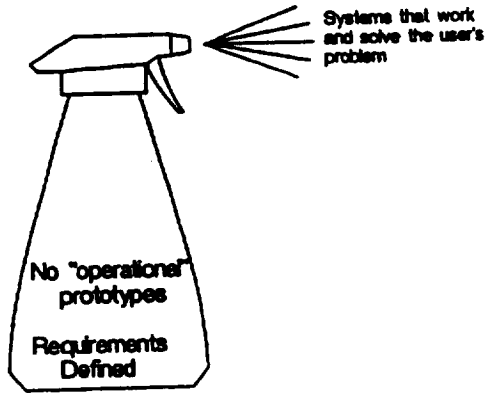
- Consider using an independent organization for final V&V
  - » A "fresh look" can often find additional errors
- Use test techniques that find errors as early as possible
- Do not forget to do regression test
  - » Easier when following a "test as you go" approach
- Prioritize the test approach
  - » Focus on critical functions first
  - » Test others later as resources permit

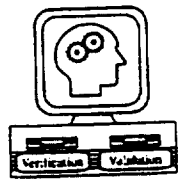
Conclusion

Doing the right things will produce the right results









Conclusion ...

Just remember

- **Software engineering is not easy**

"Software engineering is harder than you think. I can not emphasize strongly enough how true this statement is."9

- **Expert Systems are software**

"AI entails massive software engineering."9

They do not work "like magic"

References

1. Behrendt, W., Lambert, S., and Ringland, G.. "An Outline Model for Reasoning about KBS Projects and Development Risks". *Heuristics.* Volume 4 Number 4, pp. 30-38, Winter 1991.

2. Boehm, B.. "Industrial Software Metrics Top 10 List". *IEEE Software.* Volume 4 Number 5, pp. 84-85, September 1987.

3. Brooks, F.. *The Mythical Man Month.* Addison-Wesley, 1975.

4. Geissman, J.R.. "Verification and Validation for Expert Systems: A Practical Methodology". *Proceedings of the SOAR Conference.* 1990.

5. Guttag, J.V.. "Why Programming is Too Hard and What to Do About It". *Research Directions in Computer Science: An MIT Perspective.* MIT Press, 1991.

6. Kemmerer, R.A.. "Integrating Formal Methods into the Development Process". *IEEE Software.* pp. 37-50, September 1990.

7. Marcot, B.. "Testing Your Knowledge Base". *AI Expert,* July 1987.

8. Nielsen, J.. "Big Paybacks from 'Discount' Usability Engineering". *IEEE Software.* Volume 7 Number 3, pp. 107-108, May 1990.

References ...

9. Schank, R.C.. "Where's the AI?". *AI Magazine*. Winter 1991

10. Wallace, D.R. and Fujii, R.U.. "Software Verification and Validation". *IEEE Software*. Volume 6 No. 3, pp. 10-17, May 1989.